National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service

Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser a désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

NL-339 (r. 88/04) c

Canada

UNIVERSITY OF ALBERTA

FINITE ELEMENT MODELING OF OPEN CHANNEL FLOW

BY

FAYE E. HICKS

A THESIS
SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH
IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE
OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF CIVIL ENGINEERING

EDMONTON, ALBERTA
FALL, 1990

National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service

Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-64989-5

UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR:	FAYE E. HICKS
TITILE OF THESIS:	FINITE ELEMENT MODELING OF OPEN CHANNEL FLOW
DEGREE:	DOCTOR OF PHILOSOPHY

YEAR THIS DEGREE GRANTED: FALL 1990

Permission is hereby granted to THE UNIVERSITY OF ALBERTA LIBRARY to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves other publication rights, and neither the thesis or extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

PERMANENT ADDRESS:

Box 2, Site 38, RR #4
Sherwood Park, Alberta
Canada, T8A 3K4

DATED: [illegible]

UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled FINITE ELEMENT MODELING OF OPEN CHANNEL FLOW submitted by FAYE E. HICKS in partial fulfillment of the requirements for the degree of DOCTOR OF PHILOSOPHY.

P. M. STEFFLER, SUPERVISOR

N. RAJARATNAM

R. GERARD

D. J. WILSON

N. D. KATOPODES, EXTERNAL EXAMINER

DATE: Oct 11/90

ABSTRACT

Flood forecasting and delineation; the design of flood protection works; and the assessment of the impact of dam failure or a sudden ice jam release, are all examples where knowledge of the relationships between depth and discharge in an open channel are required. However, even for a one-dimensional problem, the governing non-linear system of partial differential equations (known as the *St. Venant* equations) are extremely difficult to solve. This study provides a comprehensive examination of the state of the art in finite element modeling of open channel flow. The objective was to determine if a numerical technique exists, or could be developed, which would provide accurate and stable solutions for a wide range of one-dimensional open channel flow problems.

The finite element method was used for this study because of its underlying consistency and generality which allows for quick comparisons between numerical schemes. In addition, its geometric flexibility leads to efficient definition of the irregular features common to natural channels and boundary conditions are easily implemented. The schemes investigated were the Taylor-Galerkin; the Dissipative-Galerkin (DG); and the Least-Squares methods. In addition, a reconsideration of the fundamental role of the characteristics in the determination of the upwind weighting, has lead to a new approach entitled the Characteristic-Dissipative-Galerkin (CDG) scheme. Results were compared with the Bubnov-Galerkin and a four-point implicit finite difference scheme. Initially, a linear stability analysis was used to examine the amplification and phase characteristics of these methods. They were then tested extensively for both steady and unsteady flow situations including kinematic , diffusive and shallow water gravity waves.

By employing upwinded elements both the DG and CDG schemes were able to achieve superior performance compared to other methods, propagating disturbances

without significant damping of the peak and displaying good phase accuracy. Both methods proved stable near shocks, although the CDG did not require an increase in the upwinding parameter.

It was concluded that the success of the CDG scheme in modeling a wide variety of problems without significant parameter variation, was due to its consideration of the fundamental role of the characteristics in quantifying the upwind weighting. This method could be extended to other hyperbolic systems such as: two-dimensional flows; multi-layer fluids; or sediment transport problems.

ACKNOWLEDGEMENTS

The author wishes to acknowledge the encouragement and support provided by her supervisor, Dr. P. Steffler. His keen interest and patient guidance over the course of this project has been greatly appreciated. Also, the time and effort extended by Dr. R. Gerard in advising on the application of the model to the Hay River is also gratefully acknowledged, as is the technical support provided by Mr. S. Lovell and Mr. M. Jasek. Thanks are also extended to Dr. N. Rajaratnam and Dr. D. Wilson for taking time to review this work and to Dr. N. Katopodes for agreeing to be the external examiner.

This thesis was financially supported through scholarships from: the Natural Sciences and Engineering Research Council of Canada; the Alberta Heritage Scholarship Foundation; the University of Alberta (Faculty of Graduate Studies); the North American Life Assurance Co. with the Canadian Council of Professional Engineers; and the Province of Alberta. All are most gratefully acknowledged.

Finally, special thanks are extended to my husband, Les, and my parents Bernice and Bill Yeomans, for their continuing support and encouragement through many years of studies.

TABLE OF CONTENTS

1. INTRODUCTION 1

2. EQUATIONS OF OPEN CHANNEL FLOW 4
    2.1 INTRODUCTION 4
    2.2 EQUATIONS OF ONE-DIMENSIONAL OPEN CHANNEL FLOW 5
        2.2.1 Introduction 5
        2.2.2 Derivation of the St. Venant Equations 5
        2.2.3 Consideration of Two-Dimensional Flow 17
        2.2.4 Other Forms of the St. Venant Equations 19
    2.3 APPROXIMATIONS TO THE ST. VENANT EQUATIONS 22
        2.3.1 Introduction 22
        2.3.2 Scaling the St. Venant Equations 22
        2.3.3 The Kinematic Wave 25
        2.3.4 The Diffusive Wave 27
        2.3.5 The Shallow Water Gravity Wave 29
    2.4 ANALYTIC SOLUTIONS OF THE ST. VENANT EQUATIONS 30
        2.4.1 Introduction 30
        2.4.2 Classification of the St. Venant Equations 31
        2.4.3 The Method of Characteristics 31
        2.4.4 Linearization of the St. Venant Equations 33
        2.4.5 Boundary and Initial Conditions 37

3. NUMERICAL SOLUTION OF THE EQUATIONS OF ONE-DIMENSIONAL OPEN CHANNEL FLOW 43
    3.1 INTRODUCTION 43
    3.2 THE METHOD OF CHARACTERISTICS 44
    3.3 FINITE DIFFERENCE METHODS 45
        3.3.1 Introduction 45
        3.3.2 Upwind Finite Differences 46
        3.3.3 The Box Finite Difference Scheme 48
    3.4 FINITE ELEMENT METHODS 54

3.4.1 Introduction 54
3.4.1.1 Implementation for the Linear Convection Equation 54
3.4.1.2 Implementation for the Unsteady St. Venant Equations 58
3.4.2 The Bubnov-Galerkin Method 61
3.4.3 The Taylor-Galerkin Method 63
3.4.4 The Petrov-Galerkin Method 68
3.4.4.1 Introduction 68
3.4.4.2 The Dissipative-Galerkin Method 70
3.4.4.3 The Characteristic-Dissipative-Galerkin Method 71
3.4.4.4 Comparison of Upwinding Matrices for the Petrov-Galerkin Method 73
3.4.4.5 Choice of Upwinding Parameter for the Petrov-Galerkin Method 75
3.4.5 The Least-Squares Method 76
3.4.6 Other Finite Element Methods 79

4. LINEAR STABILITY ANALYSIS 89
4.1 INTRODUCTION 89
4.1.1 Background 89
4.1.2 Implementation 90
4.2 SHALLOW WATER GRAVITY WAVES 94
4.2.1 Introduction 94
4.2.2 The Box Finite Difference Scheme 94
4.2.3 The Bubnov-Galerkin Method 96
4.2.4 The Taylor-Galerkin Method 98
4.2.5 The Petrov-Galerkin Method 99
4.2.5.1 The Dissipative-Galerkin Method 99
4.2.5.2 The Characteristic-Dissipative-Galerkin Method 100
4.2.6 The Least-Squares Method 101
4.2.7 Comparison of Methods 102
4.3 KINEMATIC WAVES 105
4.3.1 Introduction 105
4.3.2 Kinematic Wave Analysis 106

5. NUMERICAL ANALYSIS 159
  5.1 INTRODUCTION 159
  5.2 STEADY FLOW TESTS 160
    5.2.1 Introduction 160
    5.2.2 Hydraulic Jump 161
      5.2.2.1 General 161
      5.2.2.2 Discussion of Results 161
      5.2.2.3 Spectral Analysis 165
    5.2.3 Transition from a Mild to a Steep Channel 166
      5.2.3.1 General 166
      5.2.3.2 Discussion of Results 167
    5.2.4 Comparison of Methods 168
  5.3 UNSTEADY FLOW TESTS 168
    5.3.1 Introduction 168
    5.3.2 Disturbance Propagation in a Horizontal Frictionless Flow 169
      5.3.2.1 General 169
      5.3.2.2 Linear Perturbation - Froude Number = 0.0 170
      5.3.2.3 Non-Linear Perturbation - Froude Number = 0.0 174
      5.3.2.4 Non-Linear Perturbation - Froude Number = 0.5 177
      5.3.2.5 Non-Linear Perturbation - Froude Number = 1.0 181
      5.3.2.6 Non-Linear Perturbation - Froude Number = 2.0 183
      5.3.2.7 Discussion of Results 185
    5.3.3 Dam Break Analysis 186
      5.3.3.1 General 186
      5.3.3.2 Dam Break - H1/Ho = 0.5 186
      5.3.3.3 Dam Break - H1/Ho = 0.05 189
      5.3.3.4 Discussion of Results 191
    5.3.4 Propagation of a Wave Through an Hydraulic Jump 192
      5.3.4.1 Introduction 192
      5.3.4.2 Discussion of Results 192
    5.3.5 Diffusive and Kinematic Waves 193
      5.3.5.1 Introduction 193
      5.3.5.2 Results for β = 5.5 194
      5.3.5.3 Results for β = 61 195
      5.3.5.4 Results for β = 117 197

5.3.6 Comparison of Methods 197
5.4 SYMMETRIC AND NON-CONSERVATION FORMULATIONS FOR THE CDG SCHEME 199
5.4.1 Introduction 199
5.4.2 Discussion of Results 199

6. HAY RIVER ANALYSIS 316
6.1 INTRODUCTION 316
6.1.1 General 316
6.1.2 Background 316
6.1.3 Geometric Data 317
6.1.4 Discretization 318
6.1.5 Boundary and Initial Conditions 318
6.2 STEADY FLOW ANALYSIS 319
6.3 UNSTEADY FLOW ANALYSIS 320
6.3.1 Introduction 320
6.3.2 Assumptions for Ice Jam Simulation 320
6.3.3 Ice Jam Surge Releases at Station 920 km 322
6.3.4 Ice Jam Surge Releases at Station 1095.6 km 323
6.3.5 Discussion of Results 324

REFERENCES 354

APPENDIX A - BFD Program Listing 363

APPENDIX B - Implementation of the FEM for the St. Venant Equations 384

APPENDIX C - FEM Program Listing 399

APPENDIX D - Coefficients in the Linear Stability Analysis 460

LIST OF TABLES

Table 5.1 Maximum negative deviations for the hydraulic jump test 162

Table 5.2 Wave peaks for the linear propagation test (semi-implicit)................ 171

Table 5.3 Maximum negative depth deviations for the linear propagation test (semi-implicit) 172

Table 5.4 Wave peaks for the non-linear propagation test for Froude number = 0.0 (semi-implicit) 175

Table 5.5 Maximum negative depth deviations for the non-linear propagation test for Froude number = 0.0 (semi-implicit) 176

Table 5.6 Wave peaks for the non-linear propagation test for Froude number = 0.5 (semi-implicit) 178

Table 5.7 Maximum negative depth deviations for the non-linear propagation test for Froude number = 0.5 (semi-implicit) 179

Table 5.8 Maximum negative progressive wave depth deviations for the dambreak test ($H_1/H_0 = 0.5$) 188

Table 5.9 Wave peaks for the diffusive wave test ($\beta = 5.5$)........................ 196

Table 5.10 Wave peaks for the diffusive wave test ($\beta = 61$) 198

LIST OF FIGURES

Figure 2.1 Definition sketch for one-dimensional open channel flow (a) cross section (b) profile 39

Figure 2.2 Forces acting on the one-dimensional control volume (a) profile; (b) plan view (resultants) 40

Figure 2.3 One-dimensional control volume for a stationary hydraulic jump 41

Figure 2.4 Method of characteristics for shallow water gravity waves (a) subcritical flow; (b) supercritical flow 42

Figure 3.1 Definition sketch for the finite difference grid for one-dimensional flow 81

Figure 3.2 The method of characteristics for the one-dimensional convection equation 82

Figure 3.3 Violation of the stability criteron in the method of characteristics 83

Figure 3.4 Definition sketch for the finite difference grid for four-point schemes 84

Figure 3.5 Definition sketch for the finite element method for one-dimensional flow 85

Figure 3.6 Definition sketch for the local coordinate, r, within an element method for one-dimensional flow 86

Figure 3.7 Variation of [**W**] with **F** for the DG scheme 87

Figure 3.8 Variation of [**W**] with **F** for the CDG scheme 88

Figure 4.1 Results of stability analysis, Box Finite Difference Scheme, $\mathbf{F} = 0.0$, $\theta = 0.5$ (progressive and regressive waves) 108

Figure 4.2 Results of stability analysis, Box Finite Difference Scheme, $\mathbf{F} = 0.5$ and 2.0, $\theta = 0.5$ (regressive wave) 109

Figure 4.3 Results of stability analysis, Box Finite Difference Scheme, $\mathbf{F} = 0.99$, $\theta = 0.5$ (regressive wave) 110

Figure 4.4 Results of stability analysis, Box Finite Difference Scheme, $\mathbf{F} = 0.0$, $\theta = 0.6$ (progressive and regressive waves) 111

Figure 4.5 Results of stability analysis, Box Finite Difference Scheme, $\mathbf{F} = 0.5$ and 2.0, $\theta = 0.6$ (regressive wave) 112

Figure 4.6 Results of stability analysis, Box Finite Difference Scheme, $\mathbf{F} = 0.99$, $\theta = 0.6$ (regressive wave) 113

Figure 4.7 Results of stability analysis, Bubnov-Galerkin Method, F = 0.0, θ = 0.5 (progressive and regressive waves) 114

Figure 4.8 Results of stability analysis, Bubnov-Galerkin Method, F = 0.5 and 2.0, θ = 0.5 (regressive wave) 115

Figure 4.9 Results of stability analysis, Bubnov-Galerkin Method, F = 0.99, θ = 0.5 (regressive wave) 116

Figure 4.10 Results of stability analysis, Bubnov-Galerkin Method, F = 0.0, θ = 1.0 (progressive and regressive waves) 117

Figure 4.11 Results of stability analysis, Bubnov-Galerkin Method, F = 0.5 and 2.0, θ = 1.0 (regressive wave) 118

Figure 4.12 Results of stability analysis, Bubnov-Galerkin Method, F = 0.99, θ = 1.0 (regressive wave) 119

Figure 4.13 Results of stability analysis, Taylor-Galerkin Method, F = 0.0, θ = 0.0 (progressive and regressive waves) 120

Figure 4.14 Results of stability analysis, Taylor-Galerkin Method, F = 0.5 and 2.0, θ = 0.0 (regressive wave) 121

Figure 4.15 Results of stability analysis, Taylor-Galerkin Method, F = 0.99, θ = 0.0 (regressive wave) 122

Figure 4.16 Results of stability analysis, Dissipative-Galerkin Method, F = 0.0, θ = 0.5 (progressive and regressive waves) 123

Figure 4.17 Results of stability analysis, Dissipative-Galerkin Method, F = 0.5 and Fr = 2.0, θ = 0.5 (regressive wave) 124

Figure 4.18 Results of stability analysis, Dissipative-Galerkin Method, F = 0.99, θ = 0.5 (regressive wave) 125

Figure 4.19 Results of stability analysis, Dissipative-Galerkin Method, F = 0.0, θ = 1.0 (progressive and regressive waves) 126

Figure 4.20 Results of stability analysis, Dissipative-Galerkin Method, F = 0.5 and 2.0, θ = 1.0 (regressive wave) 127

Figure 4.21 Results of stability analysis, Dissipative-Galerkin Method, F = 0.99, θ = 1.0 (regressive wave) 128

Figure 4.22 Results of stability analysis for ω = 0.25, 0.5 and 1.0, Characteristic-Dissipative-Galerkin Method, C = 0.5, θ = 0.5 (progressive wave) 129

Figure 4.23 Results of stability analysis for ω = 0.25, 0.5 and 1.0, Characteristic-Dissipative-Galerkin Method, F = 0.5 and 2.0, C = 0.5, θ = 0.5 (regressive wave) 130

Figure 4.24 Results of stability analysis for $\omega = 0.25$, 0.5 and 1.0, Characteristic-Dissipative-Galerkin Method, $C = 1.0$, $\theta = 0.5$ (progressive wave) 131

Figure 4.25 Results of stability analysis for $\omega = 0.25$, 0.5 and 1.0, Characteristic-Dissipative-Galerkin Method, $\mathbf{F} = 0.5$ and 2.0, $C = 1.0$, $\theta = 0.5$ (regressive wave) 132

Figure 4.26 Results of stability analysis, Characteristic-Dissipative-Galerkin Method, $\mathbf{F} = 0.0$, $\theta = 0.5$ (progressive and regressive waves) 133

Figure 4.27 Results of stability analysis, Characteristic-Dissipative-Galerkin Method, $\mathbf{F} = 0.5$ and 2.0, $\theta = 0.5$ (regressive wave) 134

Figure 4.28 Results of stability analysis, Characteristic-Dissipative-Galerkin Method, $\mathbf{F} = 0.99$, $\theta = 0.5$ (regressive wave) 135

Figure 4.29 Results of stability analysis, Characteristic-Dissipative-Galerkin Method, $\mathbf{F} = 0.0$, $\theta = 1.0$ (progressive and regressive waves) 136

Figure 4.30 Results of stability analysis, Characteristic-Dissipative-Galerkin Method, $\mathbf{F} = 0.5$ and 2.0, $\theta = 1.0$ (regressive wave) 137

Figure 4.31 Results of stability analysis, Characteristic-Dissipative-Galerkin Method, $\mathbf{F} = 0.99$, $\theta = 1.0$ (regressive wave) 138

Figure 4.32 Results of stability analysis, Characteristic-Dissipative-Galerkin Method, $\mathbf{F} = 0.0$, $\theta = 0.5$ (symmetric formulation) 139

Figure 4.33 Results of stability analysis, Characteristic-Dissipative-Galerkin Method, $\mathbf{F} = 0.5$ and 2.0, $\theta = 0.5$ (symmetric formulation, regressive wave) 140

Figure 4.34 Results of stability analysis, Least-Squares Method, $\mathbf{F} = 0.0$, $\theta = 0.5$ (progressive and regressive waves) 141

Figure 4.35 Results of stability analysis, Least-Squares Method, $\mathbf{F} = 0.5$ and 2.0, $\theta = 0.5$ (regressive wave) 142

Figure 4.36 Results of stability analysis, Least-Squares Method, $\mathbf{F} = 0.99$, $\theta = 0.5$ (regressive wave) 143

Figure 4.37 Results of stability analysis, Least-Squares Method, $\mathbf{F} = 0.0$, $\theta = 1.0$ (progressive and regressive waves) 144

Figure 4.38 Results of stability analysis, Least-Squares Method, $\mathbf{F} = 0.5$ and 2.0, $\theta = 1.0$ (regressive wave) 145

Figure 4.39 Results of stability analysis, Least-Squares Method, $\mathbf{F} = 0.99$, $\theta = 1.0$ (regressive wave) 146

Figure 4.40 Comparison of results of stability analysis (progressive and regressive waves), $N = L/\Delta x = 10$, $\mathbf{F} = 0.0$, $\theta = 0.5$ for all methods except Box Finite Difference ($\theta = 0.6$) and Taylor-Galerkin ($\theta = 0.0$) 147

Figure 4.41 Comparison of results of stability analysis (regressive wave), $N = L/\Delta x = 10$, $\mathbf{F} = 0.5$, $\theta = 0.5$ for all methods except Box Finite Difference ($\theta = 0.6$) and Taylor-Galerkin ($\theta = 0.0$) 148

Figure 4.42 Comparison of results of stability analysis, $N = L/\Delta x = 4$, $\mathbf{F} = 0.5$, $\theta = 0.5$ (regressive wave) 149

Figure 4.43 Effect of increasing the upwinding coefficient, w, for the Dissipative-Galerkin Method, on the results of stability analysis, $C = 0.5$, $\mathbf{F} = 0.5$, $\theta = 0.5$ (progressive wave) 150

Figure 4.44 Effect of increasing the upwinding coefficient, w, for the Dissipative-Galerkin Method, on the results of stability analysis, $C = 0.5$, $\mathbf{F} = 0.5$, $\theta = 0.5$ (regressive wave) 151

Figure 4.45 Results of stability analysis, Box Finite Difference Scheme, $\beta = \infty$, $\theta = 0.6$ (kinematic wave) 152

Figure 4.46 Results of stability analysis, Dissipative-Galerkin Scheme, $\beta = 1000$, $F = 0.5$, $\theta = 0.5$ (kinematic wave) 153

Figure 4.47 Results of stability analysis, Characteristic-Dissipative-Galerkin Scheme, $\beta = 1000$, $\mathbf{F} = 0.5$, $\theta = 0.5$ (kinematic wave) 154

Figure 4.48 Results of stability analysis, Taylor-Galerkin Scheme, $\mathbf{F} = 0.5$, $\beta = 1000$, $\theta = 0.0$ (kinematic wave) 155

Figure 4.49 Results of stability analysis, Least-Squares Scheme, $\beta = 1000$, $\mathbf{F} = 0.5$, $\theta = 0.5$ (kinematic wave) 156

Figure 4.50 Algorithmic damping of $2\Delta x$ waves for the CDG scheme as a function of β, $\theta = 0.5$, $\mathbf{F} = 0.5$, $\omega = 0.25$ 157

Figure 4.51 Results of stability analysis, Characteristic-Dissipative-Galerkin Scheme, $\beta = 5$, $\mathbf{F} = 0.5$, $\theta = 0.5$ (kinematic wave) 158

Figure 5.1 Hydraulic jump ($t = 0.144s$), Bubnov-Galerkin method, $\theta = 0.5$, $C = 1.0$, $\Delta t = 0.0072s$ 201

Figure 5.2 Hydraulic jump ($t = 0.72s$), Bubnov-Galerkin method, $\theta = 0.5$, $C = 1.0$, $\Delta t = 0.0072s$ 202

Figure 5.3 Hydraulic jump ($t = 0.72s$), Bubnov-Galerkin method, $\theta = 1.0$, $C = 10$, $\Delta t = 0.072s$ 203

Figure 5.4 Hydraulic jump ($t = 7.2s$), Bubnov-Galerkin method, $\theta = 1.0$, $C = 10$, $\Delta t = 0.072s$ 204

Figure 5.5 Hydraulic jump ($t = 0.144s$), Taylor-Galerkin method, $\theta = 0.0$, $C = 0.5$, $\Delta t = 0.0036s$ 205

Figure 5.6 Hydraulic jump ($t = 0.72$s), Taylor-Galerkin method, $\theta = 0.0$, $C = 0.5$, $\Delta t = 0.0036$s 206

Figure 5.7 Hydraulic jump ($t = 7.2$s), Taylor-Galerkin method, $\theta = 0.0$, $C = 0.5$, $\Delta t = 0.0036$s 207

Figure 5.8 Hydraulic jump ($t = 0.144$s), Taylor-Galerkin (lumped) method, $\theta = 0.0$, $C = 0.5$, $\Delta t = 0.0036$s 208

Figure 5.9 Hydraulic jump ($t = 0.72$s), Taylor-Galerkin (lumped) method, $\theta = 0.0$, $C = 0.5$, $\Delta t = 0.0036$s 209

Figure 5.10 Hydraulic jump ($t = 7.2$s), Taylor-Galerkin (lumped) method, $\theta = 0.0$, $C = 0.5$, $\Delta t = 0.0036$s 210

Figure 5.11 Hydraulic jump ($t = 0.144$s), Dissipative-Galerkin method, $\theta = 0.5$, $C = 1.0$, $\Delta t = 0.0072$s, $\omega = 0.50$ 211

Figure 5.12 Hydraulic jump ($t = 0.72$s), Dissipative-Galerkin method, $\theta = 0.5$, $C = 1.0$, $\Delta t = 0.0072$s, $\omega = 0.50$ 212

Figure 5.13 Hydraulic jump ($t = 0.72$s), Dissipative-Galerkin method, $\theta = 1.0$, $C = 10$, $\Delta t = 0.072$s, $\omega = 0.50$ 213

Figure 5.14 Hydraulic jump ($t = 7.2$s), Dissipative-Galerkin method, $q = 1.0$, $C = 10$, $\Delta t = 0.072$s, $\omega = 0.50$ 214

Figure 5.15 Hydraulic jump ($t = 0.144$s), Dissipative-Galerkin method, $\theta = 0.5$, $C = 1.0$, $\Delta t = 0.0072$s, $\omega = 0.50 \mid U+c \mid$ 215

Figure 5.16 Hydraulic jump ($t = 0.72$s), Dissipative-Galerkin method, $\theta = 0.5$, $C = 1.0$, $\Delta t = 0.0072$s, $\omega = 0.50 \mid U+c \mid$ 216

Figure 5.17 Hydraulic jump ($t = 0.72$s), Dissipative-Galerkin method, $\theta = 1.0$, $C = 10$, $\Delta t = 0.072$s, $\omega = 0.50 \mid U+c \mid$ 217

Figure 5.18 Hydraulic jump ($t = 7.2$s), Dissipative-Galerkin method, $\theta = 1.0$, $C = 10$, $\Delta t = 0.072$s, $\omega = 0.50 \mid U+c \mid$ 218

Figure 5.19 Hydraulic jump ($t = 0.144$s), Characteristic-Dissipative-Galerkin method, $\theta = 0.5$, $C = 1.0$, $\Delta t = 0.0072$s, $\omega = 0.50$ 219

Figure 5.20 Hydraulic jump ($t = 0.72$s), Characteristic-Dissipative-Galerkin method, $\theta = 0.5$, $C = 1.0$, $\Delta t = 0.0072$s, $\omega = 0.50$ 220

Figure 5.21 Hydraulic jump ($t = 0.72$s), Characteristic-Dissipative-Galerkin method, $\theta = 1.0$, $C = 10$, $\Delta t = 0.072$s, $\omega = 0.50$ 221

Figure 5.22 Hydraulic jump ($t = 7.2$s), Characteristic-Dissipative-Galerkin method, $\theta = 1.0$, $C = 10$, $\Delta t = 0.072$s, $\omega = 0.50$ 222

Figure 5.23 Hydraulic jump ($t = 0.144$s), Characteristic-Dissipative-Galerkin method, $\theta = 0.5$, $C = 1.0$, $\Delta t = 0.0072$s, $\omega = 0.25$ 223

Figure 5.24 Hydraulic jump ($t = 0.72s$), Characteristic-Dissipative-Galerkin method, $\theta = 0.5$, $C = 1.0$, $\Delta t = 0.0072s$, $\omega = 0.25$ 224

Figure 5.25 Hydraulic jump ($t = 0.72s$), Characteristic-Dissipative-Galerkin method, $\theta = 1.0$, $C = 10$, $\Delta t = 0.072s$, $\omega = 0.25$ 225

Figure 5.26 Hydraulic jump ($t = 7.2s$), Characteristic-Dissipative-Galerkin method, $\theta = 1.0$, $C = 10$, $\Delta t = 0.072s$, $\omega = 0.25$ 226

Figure 5.27 Hydraulic jump ($t = 0.144s$), Least-Squares method, $\theta = 0.5$, $C = 1.0$, $\Delta t = 0.0072s$ 227

Figure 5.28 Hydraulic jump ($t = 0.72s$), Least-Squares method, $\theta = 0.5$, $C = 1.0$, $\Delta t = 0.0072s$ 228

Figure 5.29 Hydraulic jump ($t = 0.72s$), Least-Squares method, $\theta = 1.0$, $C = 10$, $\Delta t = 0.072s$ 229

Figure 5.30 Hydraulic jump ($t = 7.2s$), Least-Squares method, $\theta = 1.0$, $C = 10$, $\Delta t = 0.072s$ 230

Figure 5.31 Results of spectral analysis on hydraulic jump depths ($t = 7.2s$), (a) Bubnov-Galerkin method, $\theta = 1.0$, $C = 10$, $\Delta t = 0.072s$ 231

Figure 5.32 Results of spectral analysis on hydraulic jump depths ($t = 7.2s$), $\theta = 1.0$, $C = 10$, $\Delta t = 0.072s$, $\omega = 0.50$, (a) Dissipative-Galerkin method; (b) Characteristic-Dissipative-Galerkin method 232

Figure 5.33 Hydraulic jump ($t = 7.2s$), Least-Squares method, $\theta = 1.0$, $C = 10$, $\Delta t = 0.072s$ 233

Figure 5.34 Transition from a mild to a steep channel , initial conditions: (a) implicit schemes; (b) Taylor-Galerkin tests 234

Figure 5.35 Transition from a mild to a steep channel, Bubnov-Galerkin method, area specified at upstream boundary, $\theta = 1.0$, $\Delta t = 0.25s$, 1000 time steps 235

Figure 5.36 Transition from a mild to a steep channel, Taylor-Galerkin method, discharge and area specified at both boundaries, $\theta = 0.0$, $\Delta t = 0.25s$, 310 time steps 236

Figure 5.37 Transition from a mild to a steep channel, Taylor-Galerkin (lumped) method, discharge and area specified at both boundaries, $\theta = 0.0$, $\Delta t = 0.25s$, 300 time steps 237

Figure 5.38 Transition from a mild to a steep channel, Dissipative-Galerkin method, area specified at upstream boundary, $\theta = 1.0$, $\omega = 0.50$, $\Delta t = 0.25s$, 2000 time steps 238

Figure 5.39 Transition from a mild to a steep channel, Characteristic-Dissipative-Galerkin method, area specified at upstream boundary, $\theta = 1.0$, $\omega = 0.50$, $\Delta t = 0.5s$, 2000 time steps 239

Figure 5.40 Transition from a mild to a steep channel, Least-Squares method, area specified at upstream boundary, $\theta = 1.0$, $\Delta t = 0.125s$, 300 time steps 240

Figure 5.41 Initial conditions for linear test, $\mathbf{F} = 0.0$ 241

Figure 5.42 Figure 5.42 Linear perturbation (t = 25.6s), Box Finite Difference method, $\mathbf{F} = 0.0$, $\theta = 0.5$, (a) $C = 1.0$, $\Delta t = 0.8533s$; (b) $C = 0.5$, $\Delta t = 0.42665s$ 242

Figure 5.43 Linear perturbation (t = 25.6s), Bubnov-Galerkin method, $\mathbf{F} = 0.0$, $\theta = 0.5$ (a) $C = 1.0$, $\Delta t = 0.8533s$; (b) $C = 0.5$, $\Delta t = 0.42665s$ 243

Figure 5.44 Linear perturbation (t = 25.6s), Taylor-Galerkin method, $\mathbf{F} = 0.0$, $\theta = 0.0$, (a) $C = 0.5$, $\Delta t = 0.42665s$; (b) $C = 0.25$, $\Delta t = 0.213325s$ 244

Figure 5.45 Linear perturbation (t = 25.6s), Taylor-Galerkin (lumped) method, $\mathbf{F} = 0.0$, $\theta = 0.0$ (a) $C = 0.5$, $\Delta t = 0.42665s$; (b) $C = 0.25$, $\Delta t = 0.213325s$ 245

Figure 5.46 Linear perturbation (t = 25.6s), Dissipative-Galerkin method, $\mathbf{F} = 0.0$, $\theta = 0.5$, $\omega = 0.25$, (a) $C = 1.0$, $\Delta t = 0.8533s$; (b) $C = 0.5$, $\Delta t = 0.42665s$ 246

Figure 5.47 Linear perturbation (t = 25.6s), Characteristic-Dissipative-Galerkin method, $\mathbf{F} = 0.0$, $\theta = 0.5$, $\omega = 0.25$, (a) $C = 1.0$, $\Delta t = 0.8533s$; (b) $C = 0.5$, $\Delta t = 0.42665s$ 247

Figure 5.48 Linear perturbation (t = 25.6s), Least-Squares method, $\mathbf{F} = 0.0$, $\theta = 0.5$, (a) $C = 1.0$, $\Delta t = 0.8533s$; (b) $C = 0.5$, $\Delta t = 0.42665s$ 248

Figure 5.49 Initial conditions for non-linear test, $\mathbf{F} = 0.0$ 249

Figure 5.50 Non-linear perturbation (t = 17.3s), Box Finite Difference method, $\mathbf{F} = 0.0$, $\theta = 0.6$, (a) $C = 1.0$, $\Delta t = 0.5780s$; (b) $C = 0.5$, $\Delta t = 0.2890s$ 250

Figure 5.51 Non-linear perturbation (t = 17.3s), Bubnov-Galerkin method, $\mathbf{F} = 0.0$, $\theta = 0.5$ (a) $C = 1.0$, $\Delta t = 0.5780s$; (b) $C = 0.5$, $\Delta t = 0.2890s$ 251

Figure 5.52 Non-linear perturbation (t = 17.3s), Taylor-Galerkin method, $\mathbf{F} = 0.0$, $\theta = 0.0$, (a) $C = 0.5$, $\Delta t = 0.2890s$; (b) $C = 0.25$, $\Delta t = 0.1445s$ 252

Figure 5.53 Non-linear perturbation (t = 17.3s), Taylor-Galerkin (lumped) method, $\mathbf{F} = 0.0$, $\theta = 0.0$ (a) $C = 0.5$, $\Delta t = 0.2890s$; (b) $C = 0.25$, $\Delta t = 0.1445s$ 253

Figure 5.54 Non-linear perturbation ($t = 17.3$s), Dissipative-Galerkin method, $\mathbf{F} = 0.0$, $\theta = 0.5$, $\omega = 0.25$, (a) $C = 1.0$, $\Delta t = 0.5780$s; (b) $C = 0.5$, $\Delta t = 0.2890$s 254

Figure 5.55 Non-linear perturbation ($t = 17.3$s), Characteristic-Dissipative-Galerkin method, $\mathbf{F} = 0.0$, $\theta = 0.5$, $\omega = 0.25$, (a) $C = 1.0$, $\Delta t = 0.5780$s; (b) $C = 0.5$, $\Delta t = 0.2890$s 255

Figure 5.56 Non-linear perturbation ($t = 17.3$s), $\mathbf{F} = 0.0$, $\theta = 0.5$, $C = 1.0$, $\Delta t = 0.5780$s, (a) Dissipative-Galerkin method, $\omega = 0.50\,| U + c |$; (b) Characteristic-Dissipative-Galerkin method, $\omega = 0.50$.......... 256

Figure 5.57 Non-linear perturbation ($t = 17.3$s), Least-Squares method, $\mathbf{F} = 0.0$, $\theta = 0.5$, (a) $C = 1.0$, $\Delta t = 0.5780$s; (b) $C = 0.5$, $\Delta t = 0.2890$s 257

Figure 5.58 Initial conditions for non-linear test, $\mathbf{F} = 0.5$ 258

Figure 5.59 Non-linear perturbation ($t = 12.95$s), Box Finite Difference method, $\mathbf{F} = 0.5$, $\theta = 0.6$, (a) $C = 1.0$, $\Delta t = 0.4317$s; (b) $C = 0.5$, $\Delta t = 0.2159$s.......... 259

Figure 5.60 Non-linear perturbation ($t = 12.95$s), Bubnov-Galerkin method, $\mathbf{F} = 0.5$, $\theta = 0.5$ (a) $C = 1.0$, $\Delta t = 0.4317$s; (b) $C = 0.5$, $\Delta t = 0.2159$s 260

Figure 5.61 Non-linear perturbation ($t = 12.95$s), Taylor-Galerkin method, $\mathbf{F} = 0.5$, $\theta = 0.0$, (a) $C = 0.5$, $\Delta t = 0.2159$s; (b) $C = 0.25$, $\Delta t = 0.1079$s 261

Figure 5.62 Non-linear perturbation ($t = 12.95$s), Taylor-Galerkin (lumped) method, $\mathbf{F} = 0.5$, $\theta = 0.0$ (a) $C = 0.5$, $\Delta t = 0.2159$s; (b) $C = 0.25$, $\Delta t = 0.1079$s 262

Figure 5.63 Non-linear perturbation ($t = 12.95$s), Dissipative-Galerkin method, $\mathbf{F} = 0.5$, $\theta = 0.5$, $\omega = 0.25$, (a) $C = 1.0$, $\Delta t = 0.4317$s; (b) $C = 0.5$, $\Delta t = 0.2159$s 263

Figure 5.64 Non-linear perturbation ($t = 12.95$s), Characteristic-Dissipative-Galerkin method, $\mathbf{F} = 0.5$, $\theta = 0.5$, $\omega = 0.25$, (a) $C = 1.0$, $\Delta t = 0.4317$s; (b) $C = 0.5$, $\Delta t = 0.2159$s.......... 264

Figure 5.65 Non-linear perturbation ($t = 12.95$s), $\mathbf{F} = 0.5$, $\theta = 0.5$, $C = 1.0$, $\Delta t = 0.4317$s, (a) Dissipative-Galerkin method, $\omega = 0.50\,| U + c |$; (b) Characteristic-Dissipative-Galerkin method, $\omega = 0.50$.......... 265

Figure 5.66 Non-linear perturbation ($t = 12.95$s), Least-Squares method, $\mathbf{F} = 0.5$, $\theta = 0.5$, (a) $C = 1.0$, $\Delta t = 0.4317$s; (b) $C = 0.5$, $\Delta t = 0.2159$s 266

Figure 5.67 Initial conditions for non-linear test, $\mathbf{F} = 1.0$ 267

Figure 5.68 Non-linear perturbation ($t = 11.0$s), Box Finite Difference method, $\mathbf{F} = 1.0$, $\theta = 0.5$, (a) $C = 1.0$, $\Delta t = 0.3670$s; (b) $C = 0.5$, $\Delta t = 0.1835$s 268

Figure 5.69 Non-linear perturbation ($t = 11.0$s), Bubnov-Galerkin method, $\mathbf{F} = 1.0$, $\theta = 0.5$ (a) $C = 1.0$, $\Delta t = 0.3670$s; (b) $C = 0.5$, $\Delta t = 0.1835$s 269

Figure 5.70 Non-linear perturbation ($t = 11.0$s), Taylor-Galerkin method, $\mathbf{F} = 1.0$, $\theta = 0.0$, (a) $C = 0.5$, $\Delta t = 0.1835$s; (b) $C = 0.25$, $\Delta t = 0.09175$s 270

Figure 5.71 Non-linear perturbation ($t = 11.0$s), Taylor-Galerkin (lumped) method, $\mathbf{F} = 0.0$, $\theta = 0.5$ (a) $C = 0.5$, $\Delta t = 0.1835$s; (b) $C = 0.25$, $\Delta t = 0.09175$s 271

Figure 5.72 Non-linear perturbation ($t = 11.0$s), Dissipative-Galerkin method, $\mathbf{F} = 1.0$, $\theta = 0.5$, $\omega = 0.25$, (a) $C = 1.0$, $\Delta t = 0.3670$s; (b) $C = 0.5$, $\Delta t = 0.1835$s 272

Figure 5.73 Non-linear perturbation ($t = 11.0$s), $\mathbf{F} = 1.0$, $\theta = 0.5$, $C = 1.0$, $\Delta t = 0.3670$s, (a) Dissipative-Galerkin method, $\omega = 0.50 \mid U + c \mid$; (b) Characteristic-Dissipative-Galerkin method, $\omega = 0.50$ 273

Figure 5.74 Non-linear perturbation, Least-Squares method, $\mathbf{F} = 1.0$, $\theta = 0.5$, (a) $C = 1.0$, $\Delta t = 0.3670$s, $t = 1.468$s; (b) $C = 0.5$, $\Delta t = 0.1835$s, $t = 11.0$s 274

Figure 5.75 Initial conditions for non-linear test, $\mathbf{F} = 2.0$ 2756

Figure 5.76 Non-linear perturbation ($t = 7.7$s), Box Finite Difference method, $\mathbf{F} = 2.0$, $\theta = 0.5$, (a) $C = 1.0$, $\Delta t = 0.2566$s; (b) $C = 0.5$, $\Delta t = 0.1283$s 276

Figure 5.77 Non-linear perturbation, Bubnov-Galerkin method, $\mathbf{F} = 2.0$, $\theta = 0.5$ (a) $C = 1.0$, $t = 5.13$s, $\Delta t = 0.2566$s; (b) $C = 0.5$, $t = 7.7$s, $\Delta t = 0.1283$s 277

Figure 5.78 Non-linear perturbation ($t = 7.7$s), Taylor-Galerkin method, $\mathbf{F} = 2.0$, $\theta = 0.0$, (a) $C = 0.5$, $\Delta t = 0.1283$s; (b) $C = 0.25$, $\Delta t = 0.06415$s 278

Figure 5.79 Non-linear perturbation ($t = 7.7$s), Taylor-Galerkin (lumped) method, $\mathbf{F} = 2.0$, $\theta = 0.5$ (a) $C = 0.5$, $\Delta t = 0.1283$s; (b) $C = 0.25$, $\Delta t = 0.06415$s 279

Figure 5.80 Non-linear perturbation ($t = 7.7$s), Dissipative-Galerkin method, $\mathbf{F} = 2.0$, $\theta = 0.5$, $\omega = 0.25$, (a) $C = 1.0$, $\Delta t = 0.2566$s; (b) $C = 0.5$, $\Delta t = 0.1283$s 280

Figure 5.81 Non-linear perturbation ($t = 7.7s$), Characteristic-Dissipative-Galerkin method, $\mathbf{F} = 2.0$, $\theta = 0.5$, $\omega = 0.25$, (a) $C = 1.0$, $\Delta t = 0.2566s$; (b) $C = 0.5$, $\Delta t = 0.1283s$ 281

Figure 5.82 Non-linear perturbation ($t = 7.7s$), $\mathbf{F} = 2.0$, $\theta = 0.5$, $C = 1.0$, $\Delta t = 0.2566s$, (a) Dissipative-Galerkin method, $\omega = 0.50 \mid U + c \mid$; (b) Characteristic-Dissipative-Galerkin method, $\omega = 0.50$ 282

Figure 5.83 Non-linear perturbation ($t = 3.34s$), Least-Squares method, $\mathbf{F} = 2.0$, $\theta = 0.5$, (a) $C = 1.0$, $\Delta t = 0.2566s$; (b) $C = 0.5$, $\Delta t = 0.1283s$ 283

Figure 5.84 Dambreak, $H_1/H_o = 0.5$, ($t = 0s$), initial conditions 284

Figure 5.85 Dambreak, $H_1/H_o = 0.5$, ($t = 60s$), Box Finite Difference method, $\theta = 0.6$: (a) $\Delta t = 1.25s$; (b) $\Delta t = 2.50s$ 285

Figure 5.86 Dambreak, $H_1/H_o = 0.5$, ($t = 60s$), Bubnov-Galerkin method, $\Delta t = 1.25s$: (a) $\theta = 0.5$; (b) $\theta = 1.0$ 286

Figure 5.87 Dambreak, $H_1/H_o = 0.5$, ($t = 60s$), Taylor-Galerkin method, $\theta = 0.0$, $\Delta t = 0.625s$ 287

Figure 5.88 Dambreak, $H_1/H_o = 0.5$, ($t = 60s$), Dissipative-Galerkin method, $\omega = 0.25$, $\Delta t = 1.25s$: (a) $\theta = 0.5$; (b) $\theta = 1.0$ 288

Figure 5.89 Dambreak, $H_1/H_o = 0.5$, ($t = 0.60s$), Characteristic-Dissipative-Galerkin method, $\omega = 0.25$, $\Delta t = 1.25s$: (a) $\theta = 0.5$; (b) $\theta = 1.0$ 289

Figure 5.90 Dambreak, $H_1/H_o = 0.5$, ($t = 60s$), Least-Squares method, $\Delta t = 1.25s$: (a) $\theta = 0.5$; (b) $\theta = 1.0$ 290

Figure 5.91 Dambreak, $H_1/H_0 = 0.05$, ($t = 0s$), initial conditions 291

Figure 5.92 Dambreak, $H_1/H_0 = 0.05$, ($t = 60s$), Bubnov-Galerkin method, $\Delta t = 0.625s$: (a) $\theta = 0.5$; (b) $\theta = 1.0$ 292

Figure 5.93 Dambreak, $H_1/H_0 = 0.05$, Taylor-Galerkin method, $\theta = 0.0$, (a) $\Delta t = 0.3125s$, $t = 60s$; (b) $\Delta t = 0.625s$, $t = 10s$ 293

Figure 5.94 Dambreak, $H_1/H_0 = 0.05$, Dissipative-Galerkin method, $\omega = 0.25$, $\Delta t = 0.625s$: (a) $\theta = 0.5$, $t = 55.625s$; (b) $\theta = 1.0$, $t = 60s$ 294

Figure 5.95 Dambreak, $H_1/H_0 = 0.05$, ($t = 0.60s$), Characteristic-Dissipative-Galerkin method, $\omega = 0.25$, $\Delta t = 0.625s$: (a) $\theta = 0.5$; (b) $\theta = 1.0$ 295

Figure 5.96 Dambreak, $H_1/H_0 = 0.05$, Least-Squares method, $\Delta t = 0.625s$: (a) $\theta = 0.5$, $t = 60s$; (b) $\theta = 1.0$, $t = 60s$ 296

Figure 5.97 Propagation of a wave through an hydraulic jump ($t = 0s$), initial conditions 297

Figure 5.98 Propagation of a wave through an hydraulic jump ($t = 10.43s$), Bubnov-Galerkin method, $\theta = 0.5$, $C = 0.5$, $\Delta t = 0.2086s$........ 298

Figure 5.99 Propagation of a wave through an hydraulic jump ($t = 25.03s$), Taylor-Galerkin method, $\theta = 0.0$, $C = 0.25$, $\Delta t = 0.1043s$........ 299

Figure 5.100 Propagation of a wave through an hydraulic jump ($t = 25.03s$), Dissipative-Galerkin method, $\theta = 0.5$, $C = 0.5$, $\Delta t = 0.2086s$, $\omega = 0.25$........ 300

Figure 5.101 Propagation of a wave through an hydraulic jump ($t = 25.03s$), Dissipative-Galerkin method, $\theta = 0.5$, $C = 0.5$, $\Delta t = 2086s$, $w = 0.50 \mid U+c \mid$........ 301

Figure 5.102 Propagation of a wave through an hydraulic jump ($t = 25.03s$), Characteristic-Dissipative-Galerkin method, $\theta = 0.5$, $C = 0.5$, $\Delta t = 0.2086s$, $w = 0.25$........ 302

Figure 5.103 Propagation of a wave through an hydraulic jump ($t = 25.03s$), Least-Squares method, $\theta = 0.5$, $C = 0.5$, $\Delta t = 0.2086s$........ 303

Figure 5.104 Kinematic wave test, $\beta = 5.5$, initial conditions........ 304

Figure 5.105 Figure 5.105 Kinematic wave test, $\beta = 5.5$, (a) Box Finite Difference Method, $\theta = 0.6$, $\Delta t = 360s$; (b) Bubnov-Galerkin method, $\theta = 0.5$, $\Delta t = 180s$........ 305

Figure 5.106 Kinematic wave test, $\beta = 5.5$, $\Delta t = 180s$, $\theta = 0.5$, $\omega = 0.25$, (a) Dissipative-Galerkin method; (b) Characteristic-Dissipative-Galerkin method........ 306

Figure 5.107 Kinematic wave test, $\beta = 5.5$, $\Delta t = 15s$, (a) Taylor-Galerkin method, $\theta = 0.0$; (b) Least-Squares method, $\theta = 0.5$........ 307

Figure 5.108 Kinematic wave test, $\beta = 61$, initial conditions........ 308

Figure 5.109 Kinematic wave test, $\beta = 61$, $\Delta t = 1800s$, (a) Box Finite Difference Method, $\theta = 0.6$; (b) Bubnov-Galerkin method, $\theta = 0.5$........ 309

Figure 5.110 Kinematic wave test, $\beta = 61$, $\Delta t = 1800s$, $\theta = 0.5$, $\omega = 0.25$, (a) Dissipative-Galerkin method; (b) Characteristic-Dissipative-Galerkin method........ 310

Figure 5.111 Kinematic wave test, $\beta = 117$, initial conditions........ 311

Figure 5.112 Kinematic wave test, $\beta = 117$, $\Delta t = 1800s$, $\theta = 0.6$, Box-Finite-Difference method........ 312

Figure 5.113 Non-linear perturbation (t = 17.3s), Characteristic-Dissipative-Galerkin method, **F** = 0.0, $\theta = 0.5$, $\omega = 0.25$, C = 1.0, $\Delta t = 0.578$s (a) conservation form of the equations vs. symmetric form; (b) conservation form of the equations vs. non-conservation form 313

Figure 5.114 Non-linear perturbation (t = 14.0s), Characteristic-Dissipative-Galerkin method, **F** = 0.5, $\theta = 0.5$, $\omega = 0.25$, C = 1.0, $\Delta t = 0.4317$s (a) conservation form of the equations vs. symmetric form; (b) conservation form of the equations vs. non-conservation form 314

Figure 5.115 Dambreak, $H_1/H_0 = 0.5$, (t = 60s), Characteristic-Dissipative-Galerkin method, $\theta = 0.5$, $\omega = 0.25$, $\Delta t = 1.25$s (a) conservation form of the equations vs. symmetric form; (b) conservation form of the equations vs. non-conservation form 315

Figure 6.1 Location sketch for the Hay River basin 325

Figure 6.2 Bed profile of the Hay River 326

Figure 6.3 Hay River channel widths 327

Figure 6.4 Steady gradually varied flow profile for the entire study reach, (NTS map data) 328

Figure 6.5 Steady gradually varied flow profile in the vicinity of the falls, (NTS map data) 329

Figure 6.6 Steady gradually varied flow profile for the entire study reach, (smoothed data) 330

Figure 6.7 Steady gradually varied flow profile in the vicinity of the falls, (smoothed data) 331

Figure 6.8 Schematic describing ice jam prior to failure 332

Figure 6.9 Variation in channel width over the ice jam reach (ice jam located at Station 920 km) 333

Figure 6.10 Variation in the initial discharge over the ice jam reach (ice jam located at Station 920 km) 334

Figure 6.11 Ice jam profiles for jam located at Station 920 km 335

Figure 6.12 Hydrographs at Stations 950, 1031 and 1108 km for the jam released at Station 920 km (NTS map data) 336

Figure 6.13 Hydrographs at Stations 950, 1031 and 1108 km for the jam released at Station 920 km (smoothed data) 337

Figure 6.14 Rating curves at Station 950 km for the jam released at Station 920 km, (a) NTS map data, (b) smoothed data 338

Figure 6.15 Rating curves at Station 1031 km for the jam released at Station 920 km, (a) NTS map data, (b) smoothed data 339

Figure 6.16 Rating curves at Hay River for the jam released at Station 920 km, (a) NTS map data, (b) smoothed data 340

Figure 6.17 Variation in channel width over the ice jam reach (ice jam located at Station 1095.6 km) 341

Figure 6.18 Variation in the initial discharge over the ice jam reach (ice jam located at Station 1095.6 km) 342

Figure 6.19 Ice jam profiles for jam located at Station 1095.6 km 343

Figure 6.20 Hydrographs at Stations 1098 and 1108 km for the jam released at Station 1095.6 km (NTS map data) 344

Figure 6.21 Hydrographs at Stations 1098 and 1108 km for the jam released at Station 1095.6 km (smoothed data) 345

Figure 6.22 Rating curves at Station 1098 for the jam released at Station 1095.6 km, (a) NTS map data, (b) smoothed data 346

Figure 6.23 Rating curves at Hay River for the jam released at Station 1095.6 km, (a) NTS map data, (b) smoothed data 347

LIST OF SYMBOLS

A	channel cross sectional area perpendicular to the flow
$\mathbf{A}$	convection matrix
b	channel width
B	channel top width
c	celerity, see [2.57] page 21
C	Courant number, see [3.17] page 48
C_*	non-dimensional Chezy coefficient
$\mathbf{f}_j$	interpolation (trial) functions used in the finite element method
f	any known function
$\mathbf{F}$	Froude number, see [2.73] page 25
F_τ	friction force
g	acceleration due to gravity
G	amplification factor in stability analysis
h	depth, measured down from the water surface
H	depth of flow

H_*	non-dimensional flow depth
H'	uniform flow depth
i,j,n	integer indices
k	effective bed roughness
L	length of domain
M	number of trial functions per element in the finite element method
N	number of nodes
p	hydrostatic pressure
P	wetted perimeter
$\mathbf{P}$	pressure force
q	discharge per unit width of channel; also non-dimensional discharge perturbation per unit width of channel
Q	discharge
r	longitudinal coordinate at the element level
r	algorithmic damping
S_o, S_x	longitudinal bed slope
S_f, S_{fx}	longitudinal friction slope
S_y	lateral bed slope

S_{fy}	lateral friction slope
t	time coordinate
t_*	non-dimensional time coordinate
u	instantaneous longitudinal velocity at a point
U	cross sectionally averaged longitudinal velocity (1-D flow); depth averaged longitudinal velocity (2-D flow)
U'	uniform flow velocity
U_*	non-dimensional average longitudinal velocity
V_*	longitudinal shear velocity, see [2.14] page 9
v_i	test functions used in the finite element method
V	depth averaged lateral velocity
W	weight of fluid
x	longitudinal coordinate
x_*	non-dimensional longitudinal coordinate
y	lateral coordinate, also non-dimensional depth perturbation
z	vertical coordinate
α	kinematic wave speed coefficient
β	momentum correction coefficient, see [2.9] page 8

β	coefficient of the source terms in the non-dimensional equations, see [2.74] page 25
ε	Dissipative-Galerkin upwinding coefficient; also any minute quantity
ϕ	vector of unknowns: {U}, {U,H}, or {U,V,H}
Φ^*	nodal values of solution
γ	specific weight of water
η	non-dimensional depth to the phreatic surface at an equilibrium section of a fully developed ice jam, see [6.1] page 321
θ	implicitness
Θ	phase shift angle
μ	dynamic viscosity of water; also, ice jam internal friction-porosity coefficient
σ	wavelength, see [4.7] page 91
τ_o	shear stress
ρ	mass density of water
ω	upwinding coefficient, see [3.77] page 68

1. INTRODUCTION

The understanding and prediction of the behavior of flow in open channels is important to the solution of a wide variety of practical flow problems in water resources engineering. For example, flood forecasting and delineation; the design of flood protection works, culverts, storm sewers, spillways and diversion canals; and the assessment of the impact of dam failure or a sudden ice jam release, are all examples where knowledge of the relationships between depth, discharge and velocity in an open channel are required. However, even if the flow is approximated as one-dimensional, the governing non-linear system of partial differential equations describing these problems are extremely difficult to solve. Analytical solutions may be obtained for simplified cases by neglecting certain terms in the equations or by linearization, for example, the flow may be approximated by a kinematic or diffusive wave. Unfortunately, these solutions are restricted in their applicability as more than one flow approximation may be required within the domain of concern (either spatial or temporal). As an alternative, this study provides a comprehensive examination of the state of the art in finite element modeling of open channel flow. The objective was to determine if a numerical technique exists, or could be developed, which would provide accurate and stable solutions to the one-dimensional open channel flow equations for these types of problems.

The finite element method was used for this study because of its underlying consistency and generality which make it particularly suited to the implementation and comparison of a variety of numerical schemes. In addition, the geometric flexibility provided by the FEM readily facilitates efficient definition of the irregular features common to natural channels. Also, nonuniform discretizations may be used so that elements can be concentrated in areas where gradients are large, while fewer elements may be used in areas where the flow is more uniform, allowing for optimization of computational effort.

In Chapter 2 the equations of one-dimensional unsteady open channel flow, known as the St. Venant equations, are derived and presented in conservation, non-conservation and symmetric form. A non-dimensionalization of the equations leads to the derivation of the kinematic, diffusive and shallow water gravity wave approximations. Analytical solutions by the method of characteristics and linearization are then examined.

A number of potential numerical methods for hyperbolic systems have been presented in the literature. The finite element schemes investigated in this study were the Taylor-Galerkin ; the Dissipative-Galerkin ; the Characteristic-Dissipative-Galerkin ; and the Least-Squares methods. For illustrative purposes these were compared not only with the Bubnov-Galerkin method, but also the common four-point implicit Box Finite Difference scheme. Chapter 3 presents a general discussion of the application of these numerical methods for the St. Venant equations.

Chapter 4 presents the results of a linear stability analysis, used to examine the amplification and phase characteristics of the various numerical methods on the linearized St. Venant equations. This included consideration of both fully dynamic as well as kinematic waves. In Chapter 5, the results of a series of steady and unsteady flow tests are presented for the various schemes. The steady flow tests included the case of an hydraulic jump in a horizontal, frictionless channel and the computation of gradually varied flow profiles, for combinations of mild and steep channels. The unsteady flow situations involved the linear and non-linear propagation of shallow water gravity waves for a variety of Froude numbers, a classic dambreak problem for shallow and deep tailwater depths, and the propagation of a wave through an hydraulic jump. The behavior of the various models on kinematic and diffusive wave are presented. In addition, symmetric and non-symmetric non-conservation formulations are compared to the results obtained for the conservation form of the governing equations.

Finally, Chapter 6 presents the results of the application of the most competent method, the Characteristic-Dissipative-Galerkin scheme, to the Hay River in Alberta/Northwest Territories. This includes the determination of steady gradually varied flow profiles as well as the propagation of ice jam surges over a 500 kilometer reach.

2. EQUATIONS OF OPEN CHANNEL FLOW

2.1 INTRODUCTION

The understanding and prediction of the behavior of flow in open channels is an important aspect of water resources engineering. The design of channel erosion protection, bridges, culverts, storm sewers, spillways and diversion canals are examples where knowledge of the relationships between depth, discharge and velocity in an open channel is required. In the consideration of public safety: flood forecasting, flood zone delineation, design of flood protection works, and the impact of dam failures or a sudden ice jam release are all of concern. In addition, irrigation canals and their associated control structures present numerous practical flow problems.

Physical (scale) models are a common tool in river engineering, especially when steady flow is involved. They are particularly valuable for providing a means of visualization in complex flow situations. However, physical models have disadvantages as well. For example, because the flow depth is typically much smaller than the horizontal dimensions of a reach of interest, vertical scale distortions are often required. Also, as no practical substitute fluid for water is available, similarity criteria must be relaxed (White, 1986). Mathematical models offer an alternative to the physical modeling approach which requires no scaling. However, as will be shown, the equations represent an approximation to the true physics of the flow and must therefore be applied with due consideration for the limitations and assumptions inherent in their derivation.

2.2 EQUATIONS OF ONE-DIMENSIONAL OPEN CHANNEL FLOW

2.2.1 Introduction

The flow in an open channel is one-dimensional when variations in the dependent variables (ie. depth, discharge and velocity) occur in the longitudinal direction only (Cunge *et al*, 1980). Although one-dimensional flow rarely (if ever) exists in reality, a one-dimensional approximation is often used when the longitudinal dimension of the problem is much greater than both the stream depth and width. For example, this is done when routing a flood wave over a reach several hundred kilometers long. In general a one-dimensional approximation requires that the variations across the width and depth of the channel are negligible in comparison to the longitudinal variations (such that an average cross sectional value is representative), or that these variations can be quantified without extending the representation to higher dimensions (Cunge *et al*, 1980).

2.2.2 Derivation of the St. Venant Equations

A mathematical model of open channel flow may be developed by applying the principles of conservation of mass and momentum to a one-dimensional control volume. Figure 2.1 illustrates the notation used in this derivation. Part (a) shows a cross section where A is the flow area and B is the top width. As shown in part (b), x is the longitudinal coordinate (which follows the channel bed), while z is the depth coordinate. This assumes that the channel bed is flat with a constant slope. The flow depth is H, measured from the lowest point (thalweg) for an irregular channel section. The use of a cartesian coordinate system implies that the channel is straight and the water level across the channel is horizontal (Cunge *et al*, 1980). Therefore, deviations from these assumptions, for example in a curved channel where the water surface becomes superelevated, are neglected.

The discharge, Q, is defined as the volume of water passing through a cross section per unit time:

$$Q \equiv \int_A u \, dA \qquad [2.1]$$

where u is the longitudinal velocity at a point. The average velocity across the section, U, is defined as:

$$U \equiv \frac{Q}{A} = \frac{\int_A u \, dA}{A} \qquad [2.2]$$

Consider the control volume between sections 1 and 2 in Figure 2.1 (b). Neglecting lateral inflows, mass conservation requires that the net rate of increase of fluid mass stored within this control volume must equal the net rate of inflow of fluid mass into the control volume (Cunge *et al*, 1980), or:

$$\frac{\partial}{\partial t}\int_{x_1}^{x_2} \rho \, A \, dx = \rho \, Q_1 - \rho \, Q_2 \qquad [2.3]$$

where ρ is the density of the fluid, in this case water. Applying the fundamental theorem of calculus (Salas and Hille, 1982) this becomes:

$$\frac{\partial}{\partial t}\int_{x_1}^{x_2} \rho \, A \, dx = -\int_{x_1}^{x_2} \frac{\partial}{\partial x}(\rho \, Q) \, dx \qquad [2.4]$$

provided (ρQ) is a continuous, differentiable function.

Over the range of temperatures and velocities encountered in practical river engineering situations, the density of water is considered constant (Henderson, 1966; Streeter and Wylie, 1981). Therefore, equation [2.4] is rewritten as:

$$\frac{\partial}{\partial t}\int_{x_1}^{x_2} A\,dx + \int_{x_1}^{x_2} \frac{\partial Q}{\partial x}\,dx = 0 \qquad [2.5]$$

and since x_1 and x_2 are arbitrary:

$$\frac{\partial A}{\partial t} + \frac{\partial Q}{\partial x} = 0 \qquad [2.6]$$

provided A and Q are continuous, differentiable functions (Cunge *et al*, 1980) and the control volume location is fixed in time. Equation [2.6] represents conservation of mass for one-dimensional open channel flow and is known as the *continuity* equation.

Figure 2.2 considers a force balance on this one-dimensional control volume. Applying the principle of conservation of momentum in the x-direction, the net rate of increase of longitudinal momentum stored within this control volume must equal the net rate of inflow of longitudinal momentum into the control volume plus the net force acting on the control volume in the x-direction (Cunge *et al*, 1980).

The rate of change of momentum within the control volume is equal to the rate of change of mass times velocity (Chow, 1959):

$$\frac{\partial}{\partial t}\int_{x_1}^{x_2} \rho\, A\left(\frac{\int_A u\,dA}{A}\right)dx = \frac{\partial}{\partial t}\int_{x_1}^{x_2} \rho\left(\int_A u\,dA\right)dx = \frac{\partial}{\partial t}\int_{x_1}^{x_2} \rho\, Q\,dx \qquad [2.7]$$

The amount of momentum passing into the control volume per unit time is equal to the mass flow rate times the velocity (Chow, 1959):

$$\text{momentum entering the control volume per unit time} = \int_A \rho\, u^2\, dA = \beta\, \rho\, Q\, U \qquad [2.8]$$

where ß is known as the momentum correction (or Boussinesq) coefficient defined by:

$$ß \equiv \frac{\int_A u^2 \, dA}{U^2 A} \qquad [2.9]$$

(Chow, 1959). Therefore, the net rate of inflow of momentum into the control volume is:

$$\rho\,(ß\,Q\,U)_1 - \rho\,(ß\,Q\,U)_2 = -\int_{x_1}^{x_2} \frac{\partial}{\partial x}(\rho\,ß\,Q\,U)\,dx \qquad [2.10]$$

where again, the fundamental theorem of calculus has been applied. Equation [2.10] assumes that no longitudinal momentum is contributed by lateral inflows.

The value of ß is 1.0 for a uniform velocity distribution. For fairly straight prismatic channels, with no overbank flow, ß varies from about 1.01 to 1.12 (Chow, 1959). In this range, the effect of ß on the momentum is small and may be neglected (Chow, 1959). Assuming a uniform velocity distribution, the net rate of inflow of momentum into the control volume becomes:

$$\begin{array}{c}\text{net rate of inflow of momentum}\\ \text{into the control volume}\end{array} = -\int_{x_1}^{x_2} \frac{\partial}{\partial x}(\rho\,Q\,U)\,dx \qquad [2.11]$$

It is assumed that the only forces of consequence acting on the one-dimensional control volume, as illustrated in Figure 2.2 (a), are boundary resistance, F_τ; gravity, which is manifest in the downslope component of weight, W_x; and the pressure force, **P**. Thus the one-dimensional equation of conservation of longitudinal momentum may be written as:

$$\frac{\partial}{\partial t}\int_{x_1}^{x_2} \rho\,Q\,dx + \int_{x_1}^{x_2} \frac{\partial}{\partial x}(\rho\,Q\,U)\,dx = \mathbf{P} - F_\tau + W_x \qquad [2.12]$$

The boundary resistance may be separated into two components, one which acts on the channel bed (and banks) and one which acts on the free surface. The shear on the free surface is generally due to air resistance although wind effects may be of concern as well. In open channel flow, this component is considered negligible in comparison to the effects of bed shear (Cunge *et al*, 1980; Yevjevich, 1975; Henderson, 1966) and the resistance force is calculated as the product of the boundary shear stress, τ_o, and the surface wetted by the flow:

$$F_\tau = \int_{x_1}^{x_2} P \ \tau_o \, dx \qquad [2.13]$$

where P is the wetted perimeter of the channel cross section.

To quantify the shear stress, τ_o, it is assumed that frictional resistance formulae for steady flow are applicable to unsteady flow (Yevjevich, 1975; Henderson, 1966). Considering the shear velocity, V_* (Schlichting,1979);

$$V_* = \sqrt{\frac{\tau_o}{\rho}} \qquad [2.14]$$

and defining the friction slope S_f (Chow, 1959) using,

$$V_* = \sqrt{g \, R \, S_f} \qquad [2.15]$$

where, R is the hydraulic radius, equal to the area divided by the wetted perimeter,

$$R = \frac{A}{P} \qquad [2.16]$$

equation [2.13] may be rewritten:

$$F_\tau = \int_{x_1}^{x_2} \rho g \, A \, S_f \, dx \qquad [2.17]$$

For this study the friction slope was evaluated based on Chezy's equation,

$$U = C_* \sqrt{g\,R\,S_f} \qquad [2.18]$$

where C_* is the non-dimensional Chezy coefficient, (Henderson, 1966), C_* was determined based on the the log-law (Schlichting,1979):

$$C_* = \frac{U}{V_*} = 5.75 \log\left(\frac{R}{k}\right) + K \qquad [2.19]$$

where k is the effective channel roughness (Schlichting,1979) and K is a constant. Based on experiments and theoretical analysis, values of K for pipe flow of 6.6 and 6.5 have been determined (Keulegan, 1938), respectively. A theoretical value of 6.0 has been determined for the flow between two infinite parallel plates (Keulegan, 1938; White, 1986). Based on these, an intermediate value of $K = 6.2$ was considered appropriate for an open channel, for this study.

An alternative approach for evaluating the friction slope is to use Manning's equation (Chow, 1959),

$$U = \frac{R^{2/3}\,S^{1/2}}{n} \qquad [2.20]$$

(for SI units) where n in known as Manning's roughness.

When an ice cover is present, the resistance force is evaluated by increasing the wetted perimeter to include the water surface area on which the ice floats. In addition, a composite effective roughness is determined which incorporates the resistance effects of both the bed and the underside of the ice cover. This is discussed further in Chapter 6.

The weight of the fluid in the control volume is equal to W,

$$W = \int_{x_1}^{x_2} \gamma\, A\, dx \qquad [2.21]$$

where γ is the specific weight of water. If the bed elevation is denoted as z_o, the bed slope may be defined as:

$$S_o = -\frac{dz_o}{dx} = \sin\phi \qquad [2.22]$$

where ϕ is the angle the bed makes with the horizontal. If the bed slope is small enough so that the area perpendicular to the flow is not significantly different from the area calculated using the dimensions of the channel in the vertical plane, that is, such that $\sin\phi \approx \tan\phi$, then the downslope component of weight may be calculated:

$$W_x \approx W \sin\phi = \int_{x_1}^{x_2} \gamma\, A\, S_o\, dx \qquad [2.23]$$

For a slope of 1 in 10 ($\phi \approx 6°$), this amounts to an error of less than 0.5%. As most natural channels have substantially flatter slopes, this assumption is considered quite reasonable (Chow, 1959).

Referring back to Figure 2.1 (a), at an arbitrary elevation, z, such that $z \leq H$ the depth, h, measured down from the water surface is equal to:

$$h = H - z \qquad [2.24]$$

The corresponding width is denoted by b(z). The pressure forces, $\mathbf{P}_1$ and $\mathbf{P}_2$, acting at each end of the control volume, based on a hydrostatic pressure distribution, are quantified by:

$$\mathbf{P} = \int_{0}^{H} \gamma\, h\, b(z)\, dz \qquad [2.25]$$

Therefore, by the fundamental theorem of calculus,

$$\mathbf{P}_1 - \mathbf{P}_2 = -\int_{x_1}^{x_2} \frac{\partial}{\partial x}\left(\int_0^H \gamma\, h\, b(z)\, dz\right) dx \qquad [2.26]$$

provided **P** is a continuous, differentiable function.

The assumption of a hydrostatic pressure distribution is applicable only if the streamlines are straight and parallel, that is, if there is no acceleration of fluid particles in the plane perpendicular to the flow. Accelerations normal to the flow produce centrifugal forces which result in a non-hydrostatic pressure distribution (Chow, 1959). A hydrostatic pressure distribution may also be applied when the curvature or divergence of the streamlines is so gradual that accelerations of the flow normal to the longitudinal direction are negligible and therefore the linear hydrostatic pressure distribution closely approximates the actual pressure distribution (Chow, 1959).

Additional pressure forces are introduced when the channel width varies within the control volume. As illustrated in Figure 2.2 (b), the pressure on the channel walls has a component normal to the flow direction which is exactly balanced by the force the wall exerts on the fluid in the control volume. Over a very small length, dx, the hydrostatic pressure on the projected wall area normal to the x-direction is equal to:

$$\int_0^H \left\{\gamma \frac{\partial(b(z))}{\partial x}\, dx\, h\right\} dz \qquad [2.27]$$

This may be integrated over the length of the control volume to obtain:

$$\mathbf{P}_w = \int_{x_1}^{x_2} \left[\int_0^H \left\{ \gamma \frac{\partial(b(z))}{\partial x} h \right\} dz \right] dx \qquad [2.28]$$

It should be noted that sudden width changes would violate the assumptions associated with the use of a hydrostatic pressure distribution, as the streamlines would have significant curvature resulting in accelerations normal to the flow direction.

Combining the pressure forces, $\mathbf{P}_1$, $\mathbf{P}_2$ and $\mathbf{P}_w$, the net pressure force acting on the control volume is,

$$\mathbf{P} = -\rho g \int_{x_1}^{x_2} \left[\frac{\partial}{\partial x} \left(\int_0^H h\, b(z)\, dz \right) \right] dx + \rho g \int_{x_1}^{x_2} \left[\int_0^H \left(h \frac{\partial(b(z))}{\partial x} \right) dz \right] dx \qquad [2.29]$$

using $\gamma = \rho g$, where g is the acceleration due to gravity. Applying Liebnitz's theorem (Hildebrand,1976) to the depth integral in the first term and noting that $h(H) = 0$ and $b(0) = 0$,

$$\frac{\partial}{\partial x} \left(\int_0^H h\, b(z)\, dz \right) = \int_0^H \frac{\partial}{\partial x} (h\, b(z))\, dz = \int_0^H \left(\frac{\partial h}{\partial x} (b(z)) + h \frac{\partial(b(z))}{\partial x} \right) dz \qquad [2.30]$$

so that,

$$\mathbf{P} = -\rho g \int_{x_1}^{x_2} \left[\int_0^H \left\{ \frac{\partial h}{\partial x} (b(z)) \right\} dz \right] dx \qquad [2.31]$$

Therefore, from equation [2.24],

$$P = -\rho g \int_{x_1}^{x_2} \left[\int_0^H \left\{ \frac{\partial(H-z)}{\partial x} (b(z)) \right\} dz \right] dx$$

$$= -\rho g \int_{x_1}^{x_2} \left\{ \frac{\partial H}{\partial x} \int_0^H b(z)\, dz \right\} dx = -\rho g \int_{x_1}^{x_2} \left\{ A \frac{\partial H}{\partial x} \right\} dx \qquad [2.32]$$

since,

$$A = \int_0^H b(z)\, dz \qquad [2.33]$$

Substituting equations [2.17], [2.23] and [2.32] into [2.12], the x-direction momentum equation is then:

$$\frac{\partial}{\partial t} \int_{x_1}^{x_2} \rho\, Q\, dx + \int_{x_1}^{x_2} \frac{\partial}{\partial x} (\rho\, Q\, U)\, dx$$

$$= -\int_{x_1}^{x_2} \rho g\, A \frac{\partial H}{\partial x}\, dx - \int_{x_1}^{x_2} \rho g\, A\, S_f\, dx + \int_{x_1}^{x_2} \rho g\, A\, S_o\, dx \qquad [2.34]$$

Dividing through by ρ, equation [2.34] becomes:

$$\frac{\partial}{\partial t} \int_{x_1}^{x_2} Q\, dx + \int_{x_1}^{x_2} \frac{\partial}{\partial x} (Q\, U)\, dx$$

$$= -\int_{x_1}^{x_2} g\, A \frac{\partial H}{\partial x}\, dx - \int_{x_1}^{x_2} g\, A\, S_f\, dx + \int_{x_1}^{x_2} g\, A\, S_o\, dx \qquad [2.35]$$

and since x_1 and x_2 are arbitrary:

$$\frac{\partial Q}{\partial t} + \frac{\partial}{\partial x} (Q\, U) + g\, A \frac{\partial H}{\partial x} = g\, A\, S_o - g\, A\, S_f \qquad [2.36]$$

again, provided A an Q are continuous, differentiable functions. Equation [2.36] is called the momentum equation for one-dimensional open channel flow. The first two terms describe temporal and local acceleration, respectively. These terms are often referred to as the inertial terms (Ferrick, 1985). The third term represents the net longitudinal pressure force, while the last two reflect the effects of gravity and friction in the x-direction, respectively. These two equations, [2.6] and [2.36] are one form of the well known *St. Venant equations* (Cunge *et al*, 1980). Together with appropriate initial and boundary conditions, they define a 'model' for one-dimensional flow in open channels. Reiterating the assumptions used in this derivation, it has been assumed that:

- the flow varies only in the longitudinal direction or at the least, variations in the plane normal to the flow can be considered negligible;
- the channel is straight and has a horizontal water surface across its width;
- the channel bed is flat and has a constant slope (though it may be shown that this is not a requirement for the use of the equations);
- the velocity distribution is uniform across the channel;
- the channel width varies gradually;
- no lateral inflows occur;
- the fluid (water) has constant density;
- streamlines are straight and parallel;
- vertical accelerations are negligible and a hydrostatic pressure distribution exists;
- frictional resistance formulae for steady flow are applicable to unsteady flow;
- the bed slope is small enough that $\sin\phi \approx \tan\phi$; and
- both dependent variables are continuous differentiable functions.

It is clear that the assumptions required in the derivation of the St. Venant equations may limit their applicability to practical water resources problems. For example, the requirement that both dependent variables be continuous differentiable functions, means that the equations would not apply in the vicinity of shocks. Consequently, situations involving surges or hydraulic jumps could not be handled. Recall, however, that this was a mathematical condition imposed to obtain the differential form of the equations. Therefore, an alternative formulation might be used. For example, consider a force balance on the stationary hydraulic jump contained within the control volume depicted in Figure 2.3 (a). The equation for conservation of mass [2.3] across this jump may be written;

$$\frac{\partial}{\partial t}\int_{x_1}^{x_2} \rho\, A\, dx = \rho\, Q_1 - \rho\, Q_2 = 0 \qquad [2.37]$$

or,

$$Q_1 = Q_2 = Q \qquad [2.38]$$

while, neglecting bed slope and friction, the momentum equation [2.12] becomes,

$$\frac{\partial}{\partial t}\int_{x_1}^{x_2} \rho\, Q\, dx = \rho\,(\beta\, Q\, U)_1 - \rho\,(\beta\, Q\, U)_2 + \mathbf{P}_1 - \mathbf{P}_2 = 0 \qquad [2.39]$$

or, assuming $\beta_1 = \beta_2 = 1$,

$$\mathbf{P}_2 - \mathbf{P}_1 = \rho\, Q\,(U_1 - U_2) \qquad [2.40]$$

Equations [2.38] and [2.39] represent a discrete form of the one-dimensional continuity and momentum equations suitable for use in the vicinity of a shock. Equation [2.40], in particular, illustrates the balance between inertial and pressure forces which exists for the stationary hydraulic jump.

Another alternative, which would allow the use of the differential form of the equations [2.6] and [2.36], would be to introduce artificial diffusion to approximate a discontinuity as a continuous variation over a short distance. An example of this is illustrated in Figure 2.3 (b) for the stationary hydraulic jump.

The assumption that the velocity distribution across the channel is uniform excludes consideration of highly irregular sections or flow in compound channels (Chow, 1959). However, this and many of the other assumptions could be relaxed if more sophisticated representations of the physical laws were introduced. For example, non-hydrostatic pressure distributions (Dressler, 1978) and/or the evaluation of ß for non-uniform velocity distributions (Abbott, 1979) could be introduced.

2.2.3 Consideration of Two-Dimensional Flow

For some problems, the assumption that the flow is invariant except in the longitudinal direction may not be practical. For example, in flow through constrictions, around river training works, or in a compound channel section - the variations in the flow across the channel width are of interest. In these cases a two-dimensional model is often used (Yevjevich, 1975).

The two-dimensional approach brings in an additional (lateral) coordinate, y, which is perpendicular to the x-axis (Liggett, 1975). An additional equation is also introduced through a consideration of the conservation of momentum in the y-direction. The two-dimensional equations of open channel flow may be written as (Liggett, 1975):

$$\frac{\partial H}{\partial t} + \frac{\partial (HU)}{\partial x} + \frac{\partial (HV)}{\partial y} = 0 \qquad [2.41]$$

$$\frac{\partial (UH)}{\partial t} + \frac{\partial (U^2H)}{\partial x} + \frac{\partial (UVH)}{\partial y} + \frac{g}{2}\frac{\partial (H^2)}{\partial x} = gH\,(S_x - S_{fx}) \qquad [2.42]$$

$$\frac{\partial(VH)}{\partial t} + \frac{\partial(UVH)}{\partial x} + \frac{\partial(V^2H)}{\partial y} + \frac{g}{2}\frac{\partial(H^2)}{\partial y} = gH(S_y - S_{fy}) \qquad [2.43]$$

where V is the depth averaged lateral velocity (in the y-direction) and, S_x and S_y are the bed slopes in the longitudinal and lateral directions, respectively. Similarly, S_{fx} and S_{fy} represent the friction slopes in the longitudinal and lateral directions. Equation [2.41] is the continuity equation, while equations [2.42] and [2.43] are the x and y-direction momentum equations, respectively. The assumptions associated with the derivation of equations [2.41] to [2.43] (Liggett, 1975) are similar to, but less restrictive than, those for the one-dimensional equations:

- the flow does not vary in the vertical direction or at the least, variations through the depth can be considered negligible;
- the channel bed is flat;
- the velocity distribution is uniform over the depth;
- no seepage inflows/outflows or rainfall inflows occur;
- the fluid (water) has constant density;
- vertical accelerations are negligible and therefore a hydrostatic pressure distribution exists;
- frictional resistance formulae for steady flow are applicable to unsteady, two-dimensional flow;
- the bed slopes are small enough that sin $\phi_x \approx \tan \phi_x$; and, sin $\phi_y \approx \tan \phi_y$; and
- U, V, and H are continuous differentiable functions.

Although this thesis examines numerical solution techniques for the one-dimensional (St. Venant) equations, it was intended that the investigations include as a criteria suitability for application to the two-dimensional open channel flow equations.

2.2.4 Other Forms of the St. Venant Equations

The St. Venant equations derived above describe the flow in terms of two dependent variables: area and discharge. If the cross section may be approximated as a rectangular channel, where

$$A = BH \tag{2.44}$$

then the pressure term in equation [2.36] may be rewritten using,

$$\begin{aligned}\frac{\partial}{\partial x}\left(\frac{gAH}{2}\right) &= \frac{g}{2}\left(A\frac{\partial H}{\partial x} + H\frac{\partial A}{\partial x}\right) = \frac{gHB}{2}\frac{\partial H}{\partial x} + \frac{gH}{2}\frac{\partial HB}{\partial x}\\ &= \frac{gHB}{2}\frac{\partial H}{\partial x} + \left(\frac{gHB}{2}\frac{\partial H}{\partial x} + \frac{gH^2}{2}\frac{dB}{dx}\right)\\ &= gA\frac{\partial H}{\partial x} + \frac{gH^2}{2}\frac{dB}{dx} = gA\frac{\partial H}{\partial x} + \frac{gAH}{2B}\frac{dB}{dx}\end{aligned} \tag{2.45}$$

to obtain:

$$\frac{\partial Q}{\partial t} + \frac{\partial QU}{\partial x} + \frac{\partial}{\partial x}\left(\frac{gAH}{2}\right) - \frac{gAH}{2B}\frac{dB}{dx} = gA\,(S_o\text{-}S_f) \tag{2.46}$$

Equation [2.46] is the conservation form of the momentum equation (Liggett, 1975), for a rectangular cross section.

The momentum equation may also be written in non-conservation form using:

$$gH\frac{\partial A}{\partial x} = \frac{gA}{B}\frac{\partial A}{\partial x} = \frac{gA}{B}\left(B\frac{\partial H}{\partial x} + H\frac{dB}{dx}\right) = gA\frac{\partial H}{\partial x} + \frac{gAH}{B}\frac{dB}{dx} \tag{2.47}$$

as well as,

$$\begin{aligned}\frac{\partial(QU)}{\partial x} &= \frac{\partial(AU^2)}{\partial x} = U\frac{\partial(AU)}{\partial x} + AU\frac{\partial U}{\partial x} = U\left(\frac{\partial(AU)}{\partial x} + A\frac{\partial U}{\partial x}\right)\\ &= U\left(\frac{\partial Q}{\partial x} + A\frac{\partial U}{\partial x}\right) = U\left(\frac{\partial Q}{\partial x} + \frac{\partial(AU)}{\partial x} - U\frac{\partial A}{\partial x}\right) = 2U\frac{\partial Q}{\partial x} - U^2\frac{\partial A}{\partial x}\end{aligned} \tag{2.48}$$

and substituting into equation [2.36] to obtain:

$$\frac{\partial Q}{\partial t} + 2U\frac{\partial Q}{\partial x} + \left(gH - U^2\right)\frac{\partial A}{\partial x} = gA\left(S_o + \frac{H}{B}\frac{dB}{dx} - S_f\right) \qquad [2.49]$$

Equation [2.49] can be combined with the continuity equation,and presented in matrix form as:

$$\frac{\partial}{\partial t}\begin{Bmatrix} A \\ Q \end{Bmatrix} + \begin{bmatrix} 0 & 1 \\ \left(gH - U^2\right) & 2U \end{bmatrix}\frac{\partial}{\partial x}\begin{Bmatrix} A \\ Q \end{Bmatrix} + \begin{Bmatrix} 0 \\ gA\left(S_f - S_o - \frac{H}{B}\frac{dB}{dx}\right) \end{Bmatrix} = \{0\} \qquad [2.50]$$

or,

$$\frac{\partial\{\Phi\}}{\partial t} + [\mathbf{A}]\frac{\partial\{\Phi\}}{\partial x} + \{S\} = \{0\} \qquad [2.51]$$

where:

$$\{\Phi\} \equiv \begin{Bmatrix} A \\ Q \end{Bmatrix} \qquad [2.52]$$

is the solution vector,

$$[\mathbf{A}] \equiv \begin{bmatrix} 0 & 1 \\ \left(gH - U^2\right) & 2U \end{bmatrix} \qquad [2.53]$$

is known as the convection matrix, and,

$$\{S\} \equiv \begin{Bmatrix} 0 \\ gA\left(S_f - S_o - \frac{H}{B}\frac{dB}{dx}\right) \end{Bmatrix} \qquad [2.54]$$

represents the source terms.

Whether, or not a conservation form implies greater accuracy in the solution than a non-conservation form is not certain (Roache,1982). However, *in general* it has been found that conservation forms tend to give more accurate results (Roache, 1982). Therefore, the conservation form (equations [2.6] and [2.46]) was used for most of the tests conducted in this study. The non-conservation form [2.50] was used only to examine its comparative behavior.

Neglecting the source terms associated with width variations, an alternate form of St. Venant equations for a rectangular channel is (Henderson, 1966):

$$\frac{\partial H}{\partial t} + U\frac{\partial H}{\partial x} + H\frac{\partial U}{\partial x} = 0 \qquad [2.55]$$

$$\frac{\partial U}{\partial t} + U\frac{\partial U}{\partial x} + g\frac{\partial H}{\partial x} = g(S_o - S_f) \qquad [2.56]$$

These may be presented in a form with a symmetric convection matrix by introducing a new variable, c:

$$c = \sqrt{gH} \qquad [2.57]$$

which is the celerity of a small gravity wave. The celerity may be considered a convenient representation of depth which has the units of velocity, where

$$H = \frac{c^2}{g} \qquad [2.58]$$

and

$$dH = \frac{2c}{g}\,dc \qquad \text{or,} \qquad d(gH) = 2c\,dc \qquad [2.59]$$

Substitution of equations [2.58] and [2.59] into [2.55] and [2.56] yields the symmetric form of the St. Venant equations for a rectangular prismatic channel:

$$\frac{\partial}{\partial t}\begin{Bmatrix} 2c \\ U \end{Bmatrix} + \begin{bmatrix} U & c \\ c & U \end{bmatrix}\frac{\partial}{\partial x}\begin{Bmatrix} 2c \\ U \end{Bmatrix} + \begin{Bmatrix} 0 \\ g(S_f - S_o) \end{Bmatrix} = \{0\} \qquad [2.60]$$

Hughes *et al* (I and III, 1986), have shown that there may be some advantage to using a symmetric form when solving these types of equations numerically. This is examined further in Chapter 5.

2.3 APPROXIMATIONS TO THE ST. VENANT EQUATIONS

2.3.1 Introduction

The St. Venant equations, describe what are termed as *dynamic* waves (Henderson, 1966). They incorporate the combined effects of inertial, pressure, gravity and friction forces on the flow in an open channel. A consideration of the relative magnitudes of each of the terms in the St. Venant equations leads to the various flow approximations that have, historically, been of value in solving open channel flow problems (Miller and Cunge, 1975). Although these flow approximations only provide solutions to simplified forms of real world problems, from an academic point of view they are extremely valuable in facilitating an understanding of the physics of the flow.

2.3.2 Scaling the St. Venant Equations

The various flow approximations may be derived from the St. Venant equations for a rectangular prismatic channel,

$$\frac{\partial H}{\partial t} + U\frac{\partial H}{\partial x} + H\frac{\partial U}{\partial x} = 0 \qquad [2.55]$$

$$\frac{\partial U}{\partial t} + U\frac{\partial U}{\partial x} + g\frac{\partial H}{\partial x} = g\,(S_o - S_f) \qquad [2.56]$$

by employing a scaling process to non-dimensionalize the equations. The non-dimensional variables to be used in this analysis are:

$$x_* \equiv \frac{x}{L}\ ;\ H_* \equiv \frac{H}{Y}\ ;\ t_* \equiv \frac{t}{T}\ ;\ \text{and,}\ U_* \equiv \frac{U}{U_o} \qquad [2.61]$$

In equation [2.61], L is a length scale for the x-direction which represents the length of the reach of interest or the wavelength of a disturbance. For example, it could represent the distance to a control (rapids, a weir, etc.) or it might be based on hydrological considerations such as the spatial scope of a storm. This length scale might be 100 kilometers (km) or more for a flood routing problem; 1 km or less for the design of flood protection structures.

The depth scale, Y, could represent the normal depth, critical depth or some imposed depth condition. A practical range for Y in natural river channels, is 1 to 10 m, though higher values are possible (Kellerhals *et al*, 1972).

The velocity scale, U_o, might be derived from the discharge and the depth scale, Y, or could be based on a wave speed. The time scale, T, could be based upon hydrologic considerations, such as storm durations or the time of concentration for a basin (Linsley *et al*, 1975). Alternatively, the time scale might be derived, based upon the response time of the channel

$$T \equiv \frac{L}{U_o} \qquad [2.62]$$

The derivatives in the St. Venant equation [2.55] and [2.56] may be written in non-dimensional form based on equation [2.61]:

$$\frac{\partial H}{\partial t} = \frac{Y}{T}\frac{\partial H_*}{\partial t_*} \quad \text{and} \quad \frac{\partial H}{\partial x} = \frac{Y}{L}\frac{\partial H_*}{\partial x_*} \qquad [2.63]$$

and

$$\frac{\partial U}{\partial t} = \frac{U_o}{T}\frac{\partial U_*}{\partial t_*} \quad \text{and} \quad \frac{\partial U}{\partial x} = \frac{U_o}{L}\frac{\partial U_*}{\partial x_*} \qquad [2.64]$$

Substitution into the continuity equation [2.55], yields:

$$\frac{Y}{T}\frac{\partial H_*}{\partial t_*} + \frac{Y\,U_o}{L}\,U_*\frac{\partial H_*}{\partial x_*} + \frac{Y\,U_o}{L}\,H_*\frac{\partial U_*}{\partial x_*} = 0 \qquad [2.65]$$

Dividing through by $\frac{Y\,U_o}{L}$, the non-dimensional continuity equation becomes:

$$\frac{L}{T\,U_o}\frac{\partial H_*}{\partial t_*} + U_*\frac{\partial H_*}{\partial x_*} + H_*\frac{\partial U_*}{\partial x_*} = 0 \qquad [2.66]$$

Therefore, if

$$\frac{L}{T\,U_o} << 1 \qquad [2.67]$$

then the time derivative may be neglected (steady flow). Alternatively, if

$$T \approx \frac{L}{U_o} \qquad [2.68]$$

then all three terms in the equation are equally significant (unsteady flow).

The terms in the momentum equation may be non-dimensionalized in a similar fashion to obtain:

$$\frac{U_o}{T}\frac{\partial U_*}{\partial t_*} + U_*\frac{U_o^2}{L}\frac{\partial U_*}{\partial x_*} + g\frac{Y}{L}\frac{\partial H_*}{\partial x_*} = g\,S_o - \frac{U_*^2\,U_o^2}{H_*\,Y\,C_*^2} \qquad [2.69]$$

for a wide rectangular channel where R ≈ H (Chow, 1959), and using Chezy's equation [2.18] to quantify the friction slope. Dividing through by $\frac{U_o^2}{L}$, this becomes,

$$\frac{L}{T\,U_o}\frac{\partial U_*}{\partial t_*} + U_*\frac{\partial U_*}{\partial x_*} + \frac{g\,Y}{U_o^2}\frac{\partial H_*}{\partial x_*} = \frac{g\,L\,S_o}{U_o^2} - \frac{L\,U_*^2}{H_*\,Y\,C_*^2} \qquad [2.70]$$

Therefore, from equation [2.68], for unsteady flow

$$\frac{\partial U_*}{\partial t_*} + U_*\frac{\partial U_*}{\partial x_*} + \frac{g\,Y}{U_o^2}\frac{\partial H_*}{\partial x_*} = \frac{g\,L\,S_o}{U_o^2} - \frac{L\,U_*^2}{H_*\,Y\,C_*^2} \qquad [2.71]$$

or

$$\frac{\partial U_*}{\partial t_*} + U_* \frac{\partial U_*}{\partial x_*} + \frac{1}{\mathbf{F}^2}\frac{\partial H_*}{\partial x_*} = \beta\left(\frac{S_o C_*^2}{\mathbf{F}^2} - \frac{U_*^2}{H_*}\right) \qquad [2.72]$$

where **F** is the Froude number:

$$\mathbf{F} \equiv \frac{U_o}{\sqrt{gY}} \qquad [2.73]$$

and

$$\beta \equiv \frac{L}{Y C_*^2} \qquad [2.74]$$

Depending on the relative magnitude of the terms in equation [2.72] some may be neglected. The resulting equations, combined with the complete continuity equation, lead to the unsteady flow approximations which may be solved analytically in some cases (Henderson, 1966).

2.3.3 The Kinematic Wave

For problems involving long reaches, that is as $\beta \rightarrow \infty$, the friction and slope terms become much larger than the acceleration and pressure terms, and the St. Venant equations ([2.55] and [2.56]) reduce to:

$$\frac{\partial H}{\partial t} + \frac{\partial UH}{\partial x} = 0 \qquad [2.75]$$

and,

$$S_o = S_f \qquad [2.76]$$

This situation is known as the "kinematic wave approximation" (Henderson, 1966). It has also been described as "unsteady uniform flow" or "locally uniform flow" (Henderson, 1966).

Equation [2.76] may be expanded using Chezy's equation, to obtain:

$$S_o = \frac{U^2}{g R C_*^2} \qquad [2.77]$$

For a wide rectangular channel, R ≈ H, so that,

$$U \approx C_* \sqrt{g H S_o} \qquad [2.78]$$

Therefore, U is a function of H. Multiplying by the depth this becomes,

$$UH = C_* H^{3/2} \sqrt{g S_o} \qquad [2.79]$$

therefore,

$$\frac{\partial(UH)}{\partial x} = \frac{3}{2} C_* \sqrt{g S_o}\, H^{1/2} \frac{\partial H}{\partial x} \qquad [2.80]$$

neglecting the variation in C* with respect to x. Substitution equation [2.80] into [2.75], yields the kinematic wave equation:

$$\frac{\partial H}{\partial t} + \frac{3}{2} C_* \sqrt{g H S_o} \frac{\partial H}{\partial x} = \frac{\partial H}{\partial t} + \alpha U \frac{\partial H}{\partial x} = 0 \qquad [2.81]$$

where,

$$\alpha \equiv \frac{3}{2} \qquad [2.82]$$

when Chezy's equation is used. It should be noted that the value of α would vary, if C*(x,H) was introduced in equation [2.80].

Equation [2.81] is a convection equation for open channel flow in a wide rectangular channel. As there is no diffusion term in the equation, an observer travelling downstream at a speed (αU) would observe no change in flow depth. That is, the peak does not diminish as it propagates. Furthermore, there is only one convective velocity, therefore kinematic disturbances travel in the downstream direction only.

Because the disturbance propagates downstream at a speed defined by the depth:

$$\alpha U = \frac{3}{2} C_* \sqrt{g H S_o} \qquad [2.83]$$

larger depths propagate faster than smaller ones. As a result, the wave front of a kinematic wave steepens as it progresses downstream.

2.3.4 The Diffusive Wave

If the Froude number of the flow is small enough such that:

$$\frac{1}{F^2} \frac{\partial H_*}{\partial x_*} \sim \beta \left(\frac{S_o C_*^2}{F^2} - \frac{U_*^2}{H_*} \right) \qquad [2.84]$$

in equation [2.72], yet the inertial terms remain small in comparison:

$$\frac{\partial U_*}{\partial t_*} \sim U_* \frac{\partial U_*}{\partial x_*} << \frac{1}{F^2} \frac{\partial H_*}{\partial x_*} \sim \beta \left(\frac{S_o C_*^2}{F^2} - \frac{U_*^2}{H_*} \right) \qquad [2.85]$$

then the St. Venant equations can be approximated by:

$$\frac{\partial H}{\partial t} + U \frac{\partial H}{\partial x} + H \frac{\partial U}{\partial x} = 0 \qquad [2.55]$$

and,

$$\frac{\partial H}{\partial x} = S_o - S_f \qquad [2.86]$$

This is known as the "diffusive wave approximation" (Henderson, 1966).

Substituting Chezy's equation into equation [2.86], and again assuming $R = H$:

$$\frac{\partial H}{\partial x} = S_o - \frac{U^2}{g\,H\,C_*^2} \qquad [2.87]$$

then,

$$\frac{\partial^2 H}{\partial x^2} = -\frac{2U}{g\,H\,C_*^2}\frac{\partial U}{\partial x} + \frac{U^2}{g\,H^2\,C_*^2}\frac{\partial H}{\partial x} \qquad [2.88]$$

for constant S_o. Rearranging and simplifying this becomes:

$$\frac{\partial U}{\partial x} = -\frac{g\,H\,C_*^2}{2U}\frac{\partial^2 H}{\partial x^2} + \frac{U}{2H}\frac{\partial H}{\partial x} \qquad [2.89]$$

Substituting equation [2.89] into the continuity equation [2.55] yields:

$$\frac{\partial H}{\partial t} + U\frac{\partial H}{\partial x} - \frac{g\,H^2\,C_*^2}{2U}\frac{\partial^2 H}{\partial x^2} + \frac{U}{2}\frac{\partial H}{\partial x} = 0 \qquad [2.90]$$

or

$$\frac{\partial H}{\partial t} + \alpha\,U\frac{\partial H}{\partial x} = D\frac{\partial^2 H}{\partial x^2} \qquad [2.91]$$

where α is defined by equation [2.82] and,

$$D = \frac{g\,H^2\,C_*^2}{2U} \qquad [2.92]$$

Equation [2.91] is an advection-diffusion equation for open channel flow in a wide rectangular channel. This means that, unlike kinematic waves, diffusive waves attenuate as they propagate downstream. The rate at which this occurs is governed by the magnitude of the diffusion term *D*.

For a diffusive wave, the velocity is a function of not only the depth, but the slope of the water surface as well. This can be seen by rearranging equation [2.87]:

$$U = C \cdot \sqrt{g H \left(S_o - \partial H/_{\partial x} \right)} \qquad [2.93]$$

Because the leading edge is typically steeper than the back of a flood wave (Henderson, 1966), this causes a *loop rating curve* in which water levels associated with the rising limb of a hydrograph are lower than those for the falling stage. Furthermore, the maximum stage is not associated with the peak discharge (Henderson, 1966).

Diffusive waves are similar to kinematic waves in that they are propagated in the downstream direction only. The convective velocity (αU) is known as the "kinematic wave speed" (Henderson, 1966).

2.3.5 The Shallow Water Gravity Wave

For horizontal frictionless channels, the St.Venant equations reduce to:

$$\frac{\partial H}{\partial t} + U \frac{\partial H}{\partial x} + H \frac{\partial U}{\partial x} = 0 \qquad [2.55]$$

$$\frac{\partial U}{\partial t} + U \frac{\partial U}{\partial x} + g \frac{\partial H}{\partial x} = 0 \qquad [2.94]$$

Although some channels are horizontal, none are frictionless. Therefore, these equations represent an approximation to the St. Venant equations which is valid when β approaches zero.

The system of equations ([2.55] and [2.94]) may be diagonalized to obtain a decoupled system (Henderson, 1966):

$$\frac{\partial (U + 2c)}{\partial t} + (U + c) \frac{\partial (U + 2c)}{\partial x} = 0 \qquad [2.95a]$$

$$\frac{\partial (U - 2c)}{\partial t} + (U - c) \frac{\partial (U - 2c)}{\partial x} = 0 \qquad [2.95b]$$

Therefore, shallow water gravity waves have two convective velocities (U ± c). However, as in the case of the kinematic wave approximation, this system is purely convective. Therefore, an observer travelling with the progressive disturbance at a velocity (U + c), would observe no change in the quantity (U + 2c). The same is true for the regressive wave.

Because the speed of these disturbances is proportional to the square root of the depth, larger depths will be propagated faster than shallow ones. Therefore, like kinematic waves, the front of a shallow water gravity wave will steepen as it propagates.

2.4 ANALYTIC SOLUTIONS OF THE ST. VENANT EQUATIONS

2.4.1 Introduction

Direct integration of the complete St. Venant equations for natural rivers requires many simplifying assumptions (Yevjevich, 1975). For example, certain terms might be neglected, as described for the flow approximation discussed above, or a linearized form of the equations might be used. Although these assumptions are often overly restrictive and eliminate many practical problems from consideration (Henderson, 1966), analytical solutions provide a first level of comparison for numerical methods, as will be seen in Chapter 5. Furthermore, analytical solutions, like the flow approximations discussed above, aid in the understanding of the physics of the flow.

To describe a solution technique for these equations, some general definitions are required (Burnett, 1987). The *domain* describes the physical or temporal limits of interest, such as the reach length or the time frame concerned; the *governing equations* are the St. Venant equations, or the equations describing one of the flow approximations; and the *loading conditions* are the forces, velocities and discharges acting on or within the domain. Those loading conditions which act on the boundaries are specifically known as *boundary* *conditions*, while those prescribed at the beginning of the simulation period are called *initial conditions*.

2.4.2 Classification of the St. Venant Equations

The solution technique employed depends on the mathematical classification of the governing partial differential equations, for example: elliptic; parabolic; or hyperbolic. The differences determine (Chester, 1971):

a) the type of domain in which the problem is formulated; and

b) the type of boundary conditions prescribed on the boundary of this domain.

This system of non-linear partial differential equations,

$$\frac{\partial\{\Phi\}}{\partial t} + [\mathbf{A}]\frac{\partial\{\Phi\}}{\partial x} + \{S\} = \{0\} \qquad [2.51]$$

is classified as hyperbolic because the eigenvalues and eigenvectors of the convection matrix, [A] are real and distinct (Ames, 1977).

2.4.3 The Method of Characteristics

The method of characteristics is a solution technique for hyperbolic equations in which a partial differential equation is reduced to ordinary differential equations along *characteristic* lines or curves (Abbott, 1975).

For simplicity the method of characteristics is introduced through the simple linear convection equation,

$$\frac{\partial\psi}{\partial t} + \frac{\partial F}{\partial x} = 0 \quad \text{where,} \quad U = \frac{\partial F}{\partial\psi} \qquad [2.96]$$

where ψ is the quantity to be convected, such that,

$$\frac{\partial \psi}{\partial t} + U\frac{\partial \psi}{\partial x} = 0 \qquad [2.97]$$

For this equation;

$$\frac{D\psi}{Dt} = \frac{\partial \psi}{\partial t} + U\frac{\partial \psi}{\partial x} = 0 \quad \text{on} \quad U = \frac{dx}{dt} \qquad [2.98]$$

where *dx/dt* describes the *characteristic velocity* of propagation of the solution, ψ (Henderson, 1966).

This kinematic wave equation is a simplified analog of a hyperbolic system, where

$$\frac{DH}{Dt} = \frac{\partial H}{\partial t} + \alpha U\frac{\partial H}{\partial x} = 0 \quad \text{on,} \quad \frac{dx}{dt} = \alpha U \qquad [2.99]$$

The method of characteristics may also be extended to systems of hyperbolic equations, for example in the case of shallow water gravity waves:

$$\frac{D(U \pm 2c)}{Dt} = \frac{\partial (U \pm 2c)}{\partial t} + \begin{bmatrix} U+c & 0 \\ 0 & U-c \end{bmatrix}\frac{\partial (U \pm 2c)}{\partial x} = 0 \qquad [2.100]$$

on

$$\frac{dx}{dt} = U \pm c \qquad [2.101]$$

where, (U ± c) are the eigenvalues of the convection matrix, [**A**], for shallow water gravity waves ([2.55] and [2.94]):

$$[\mathbf{A}] \equiv \begin{bmatrix} U & H \\ g & U \end{bmatrix} \qquad [2.102]$$

(Fennema and Chaudhry,1987). Similarly, the St. Venant equations [2.53] may be diagonalized to obtain:

$$\frac{D(U \pm 2c)}{Dt} = \frac{\partial (U \pm 2c)}{\partial t} + \begin{bmatrix} U+c & 0 \\ 0 & U-c \end{bmatrix}\frac{\partial (U \pm 2c)}{\partial x} = g(S_o - S_f) \qquad [2.103]$$

on

$$\frac{dx}{dt} = U \pm c \qquad [2.101]$$

Therefore, shallow water gravity waves have the same characteristic equations as dynamic waves.

As discussed above, these characteristics trace the path of the solution. That is, the total derivatives represent the change in the dependent variables from the viewpoint of observers moving at (U + c) and (U - c) (Henderson, 1966). For dynamic and shallow water gravity waves, the dependent variables are (U ± 2c), rather than depth and discharge. The advantage of the method of characteristics is that the paths followed by these observers, through the x-t domain describe the entire solution of the unsteady flow problem (Henderson, 1966). This solution may be determined explicitly for the simplified case of a shallow water gravity wave (or a kinematic wave) in a rectangular channel, (Yevjevich, 1975), or numerically for more complex flow situations such as dynamic waves (Liggett and Cunge, 1975).

2.4.4 Linearization of the St. Venant Equations

One of the main problems in obtaining an analytical solution to the St. Venant equations is the fact that they are non-linear (Liggett, 1975). Therefore, a common analytical tool is to linearize these equations first. This may be illustrated using the St. Venant equations derived above [2.50], simplified for the case of a wide rectangular prismatic channel:

$$\frac{\partial}{\partial t}\begin{Bmatrix} H \\ UH \end{Bmatrix} + \begin{bmatrix} 0 & 1 \\ (gH - U^2) & 2U \end{bmatrix} \frac{\partial}{\partial x}\begin{Bmatrix} H \\ UH \end{Bmatrix} + \begin{Bmatrix} 0 \\ gH\,(S_{\text{[illegible]}} - S_0) \end{Bmatrix} = \{0\} \qquad [2.104]$$

where B is constant and R ≈ H. Let:

$$H = H' + \Delta H \qquad [2.105]$$

and

$$UH = U'H' + \Delta q \qquad [2.106]$$

where ΔH and Δq represent perturbations to a steady uniform flow in which H' is the normal depth and U' is the uniform flow velocity. Neglecting products of ΔH and Δq, substitution of equations [2.105] and [2.106], into [2.66] and [2.70] yields:

$$\frac{\partial \Delta H}{\partial t} + \frac{\partial \Delta q}{\partial x} = 0 \qquad [2.107]$$

for the continuity equation, and

$$\frac{\partial \Delta q}{\partial t} + 2U' \frac{\partial \Delta q}{\partial x} + \left(gH' - U'^2\right) \frac{\partial \Delta H}{\partial x} = g\, S_o\, \Delta H - \frac{1}{C_*^2} \frac{\left(U'H' + \Delta q\right)^2}{\left(H' + \Delta H\right)^2} \qquad [2.108]$$

for the momentum equation, where S_f has been defined using Chezy's equation. The denominator of the friction term in equation [2.108] may be expanded using the binomial theorem (Dull and Dull, 1951):

$$\frac{1}{C_*^2}\left(U'H' + \Delta q\right)^2 \frac{1}{(H')^2}\left(1 - \frac{2\Delta H}{H'}\right) = \frac{1}{C_*^2}\left(\frac{2U'\Delta q}{H'} - \frac{2U'^2\Delta H}{H'}\right) \qquad [2.109]$$

and S_o may be defined by the uniform flow ($S_o = S_f$), and Chezy's equation:

$$U' = C_* \sqrt{g\, H'\, S_o} \qquad [2.110]$$

such that

$$\frac{\partial \Delta q}{\partial t} + 2U' \frac{\partial \Delta q}{\partial x} + \left(gH' - U'^2\right) \frac{\partial \Delta H}{\partial x} = \frac{U'}{H'\, C_*^2}\left(-2\,\Delta q + 3\, U'\, \Delta H\right) \qquad [2.111]$$

Equations [2.107] and [2.111] may be non-dimensionalized using:

$$q = \frac{\Delta q}{U'H'} \; ; \; y = \frac{\Delta H}{H'} \qquad [2.112]$$

(in the same way as shown above for equations [2.62] and [2.68]) to obtain:

$$\frac{\partial y}{\partial t} + \frac{\partial q}{\partial x} = 0 \qquad [2.113]$$

$$\frac{\partial q}{\partial t} + 2\,\frac{\partial q}{\partial x} + \frac{\left(gH' - U'^2\right)}{U'^2}\,\frac{\partial y}{\partial x} = \frac{L}{H'\,C_*^2}\,(-2q + 3y) \qquad [2.114]$$

where T = L/U' has been used and the star subscripts have been removed from x and t for convenience of presentation. Equation [2.114] may be rewritten:

$$\frac{\partial q}{\partial t} + 2\,\frac{\partial q}{\partial x} + \left(\frac{1}{\mathbf{F}^2} - 1\right)\frac{\partial y}{\partial x} = \beta\,(-2q + 3y) \qquad [2.115]$$

where **F** is the Froude number for the steady uniform flow:

$$\mathbf{F} = \frac{U'}{\sqrt{gH'}} \qquad [2.116]$$

and β is,

$$\beta = \frac{L}{H'\,C_*^2} \qquad [2.117]$$

Therefore the linearized non-dimensional St. Venant equations may be written:

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \frac{\partial\{\phi\}}{\partial t} + \begin{bmatrix} 0 & 1 \\ \left(\frac{1}{\mathbf{F}^2} - 1\right) & 2 \end{bmatrix} \frac{\partial\{\phi\}}{\partial x} + \begin{bmatrix} 0 & 0 \\ -3\beta & 2\beta \end{bmatrix} \{\phi\} = \{0\} \qquad [2.118]$$

for

$$\{\phi\} \equiv \begin{Bmatrix} y \\ q \end{Bmatrix} \qquad [2.119]$$

For the case where β = 0, equation [2.118] reverts to the shallow water gravity wave equations. However, when β → ∞ this corresponds to a kinematic wave situation, where:

$$\frac{\partial y}{\partial t} + \frac{3}{2}\frac{\partial y}{\partial x} = 0 \qquad [2.120]$$

From equation [2.120] it is seen that a linear kinematic disturbance propagates at a constant velocity which is independent of the depth (and the Froude number). Therefore, the wave front does not steepen as it propagates downstream.

Because equation [2.118] is undefined for a Froude number of zero, it is an inconvenient form except for kinematic wave approximations. An alternate form may be derived using the celerity $\sqrt{gH'}$ as the velocity scale in equation [2.112] rather than U':

$$q = \frac{\Delta q}{\sqrt{gH'}H'} \; ; \; y = \frac{\Delta H}{H'} \qquad [2.121]$$

The resulting linearized non-dimensional equations are:

$$\frac{\partial\{\phi\}}{\partial t} + \begin{bmatrix} 0 & 1 \\ 1 - \mathbf{F}^2 & 2\mathbf{F} \end{bmatrix} \frac{\partial\{\phi\}}{\partial x} + \begin{bmatrix} 0 & 0 \\ \frac{-3\beta}{H'\mathbf{F}^2} & \frac{2\beta}{\mathbf{F}^2} \end{bmatrix} \{\phi\} = \{0\} \qquad [2.122]$$

This is also undefined for a Froude number of zero, but when $\beta = 0$ it may be written:

$$\frac{\partial\{\phi\}}{\partial t} + \begin{bmatrix} 0 & 1 \\ 1 - \mathbf{F}^2 & 2\mathbf{F} \end{bmatrix} \frac{\partial\{\phi\}}{\partial x} = \{0\} \qquad [2.123]$$

Equation [2.123] is a linear convection equation for shallow water gravity waves. As in the non-linear case, there are two convective velocities $\left(U' \pm \sqrt{gH'}\right)$ though in this case they are constant. Therefore, these waves do not steepen as they propagate.

Ponce and Simons (1977) provide analytical solutions for the non-dimensional linear equation ([2.118] and [2.123]), not only for kinematic and shallow water gravity waves, but diffusive waves as well. They include the derivation of the analytical diffusion associated with the linear diffusive wave.

2.4.5 Boundary and Initial Conditions

A consideration of the behavior of characteristics illustrates the boundary and initial conditions required to close the mathematical model of open channel flow. Consider, for simplicity, the shallow water gravity wave equations [2.95]. Figure 2.4 (a) illustrates the characteristics which define the solution at an arbitrary point O within the solution domain, for a subcritical flow from left to right. If the value of the dependent variables are known throughout the entire spatial domain at some initial time (say t = 0) then characteristics emanating from points L and R, tracing lines of constant ($U \pm 2c$) and intersecting at point O, define a solution at a later time. That is:

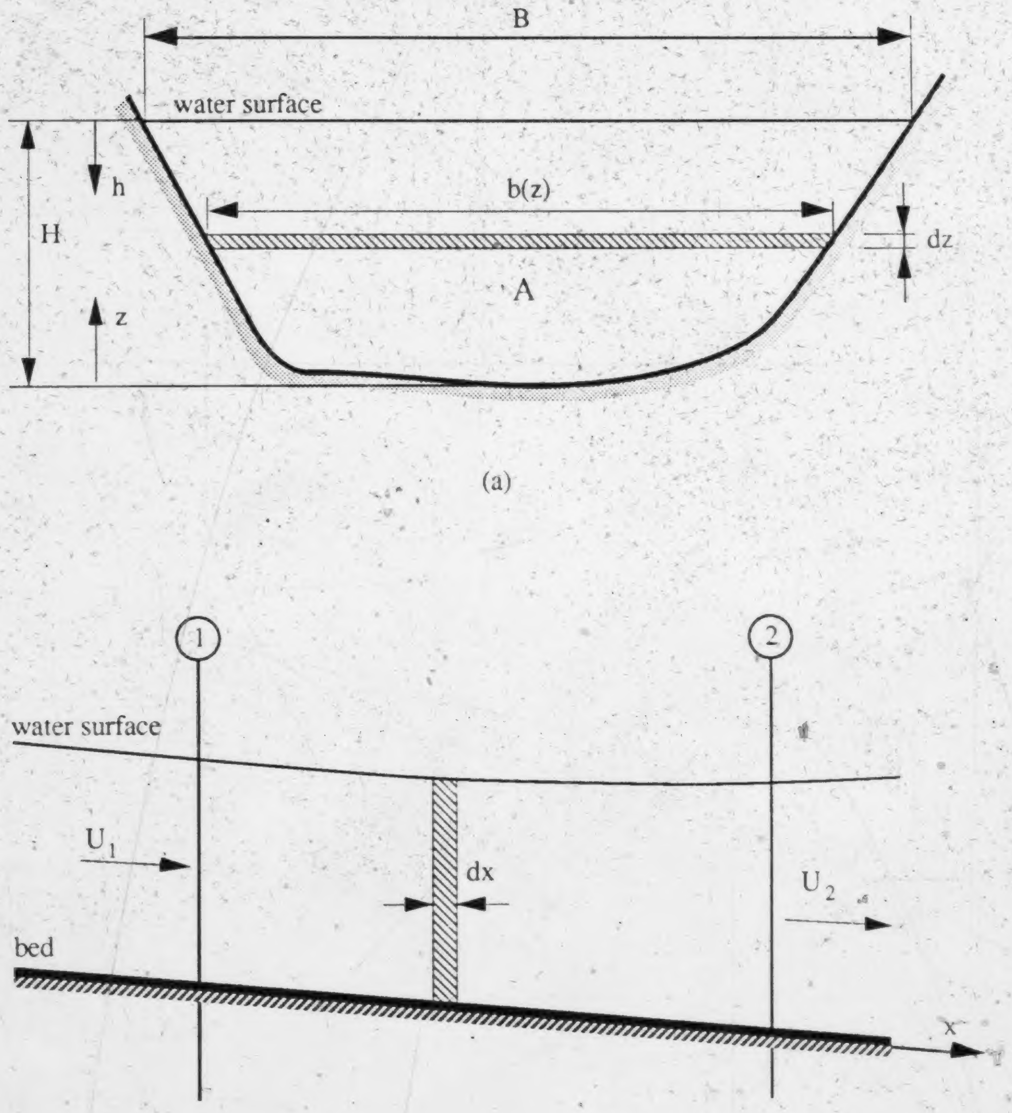
$$U_L + 2\,c_L = U_O + 2\,c_O \qquad [2.124]$$

$$U_R - 2\,c_R = U_O - 2\,c_O \qquad [2.125]$$

provides a system of two equations in two unknowns, U_O and C_O. Therefore *initial* conditions enable the determination of a solution at points within the x-t plane at some later time.

An exception to this occurs at the boundaries. For example, consider point B on the upstream boundary at x = 0. For subcritical flow, **F** < 1.0 and therefore the negative $\left(\frac{dx}{dt} = U - c\right)$ characteristic emanating from R' has a negative slope and passes through point B. However, to obtain a solution, an additional equation is required. This might consist simply of a known value of one of the independent variables, or an equation describing a relationship between the two. In either case, it is called a *boundary condition*. Similarly, a positive characteristic and a downstream boundary condition may be used at the downstream boundary. Therefore, a single boundary condition is required at each end of the spatial domain, when the flow is subcritical.

When the flow is supercritical, both characteristics have positive slopes. Therefore, as illustrated in Figure 2.4 (b), there are no characteristics emanating from the x-axis intersecting the upstream boundary at x = 0, at any time. This means that two boundary conditions are required at the upstream end of the spatial domain, when the flow is supercritical. However, there will be two characteristic equations available at the downstream boundary. Therefore, no downstream boundary conditions are required for supercritical flow.


B
water surface
h
b(z)
H
dz
A
z
1
2
water surface
U_1
dx
U_2
bed
x

(a)

(b)

Figure 2.1 Definition sketch for one-dimensional open channel flow: (a) cross section (b) profile.

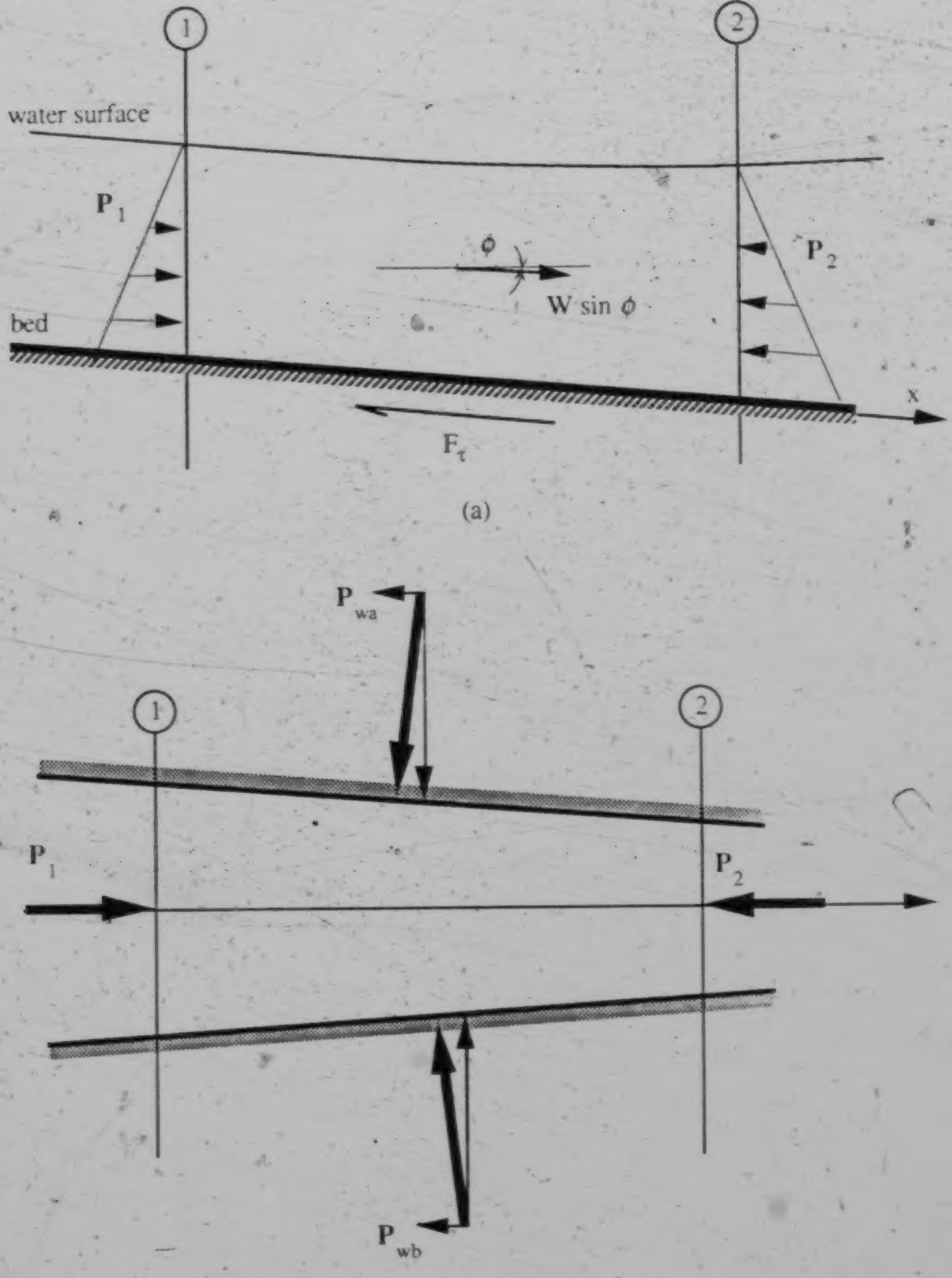
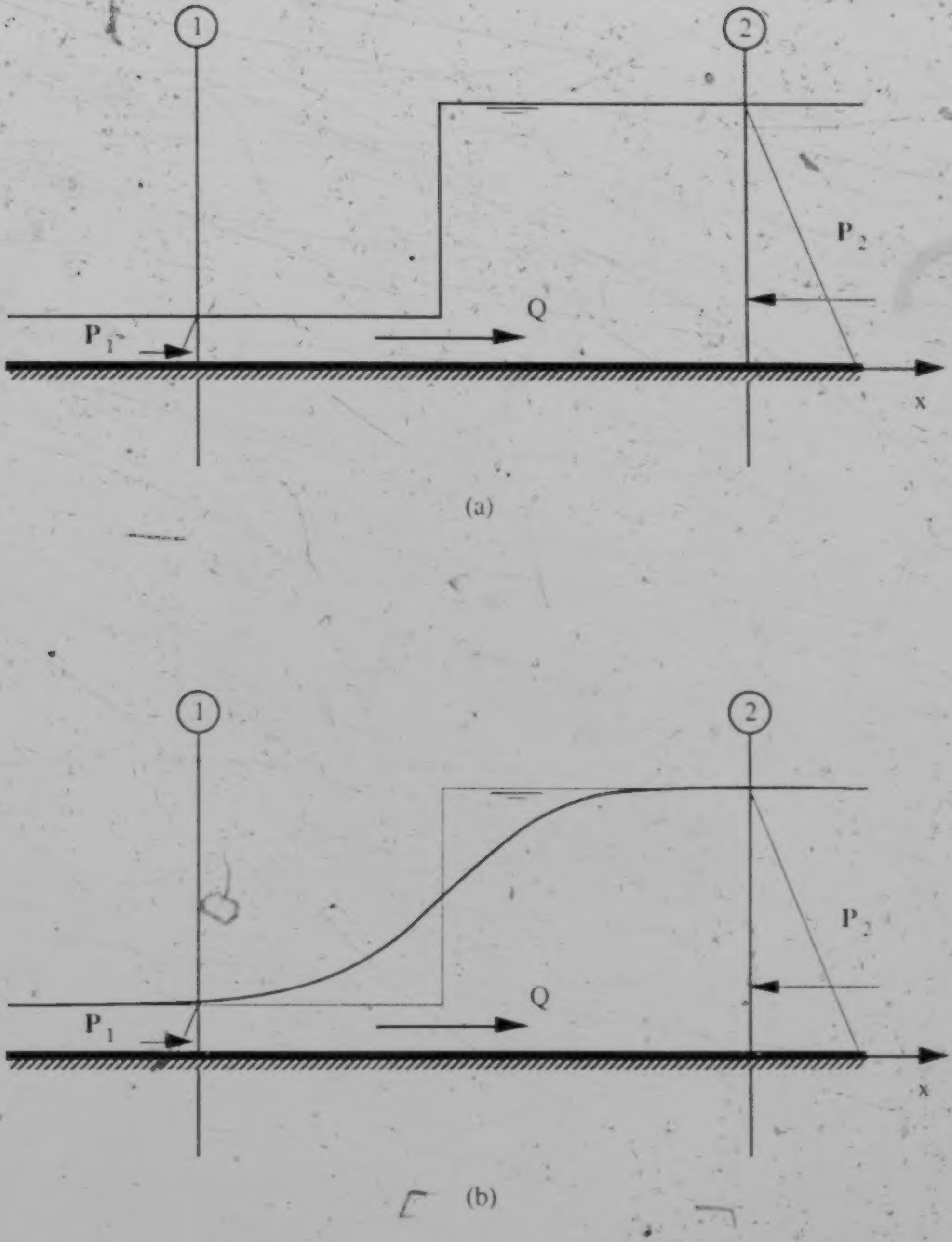

①
②
water surface
P_1
P_2
ϕ
W sin ϕ
bed
x
F_τ
(a)
P_{wa}
①
②
P_1
P_2
P_{wb}
(b)

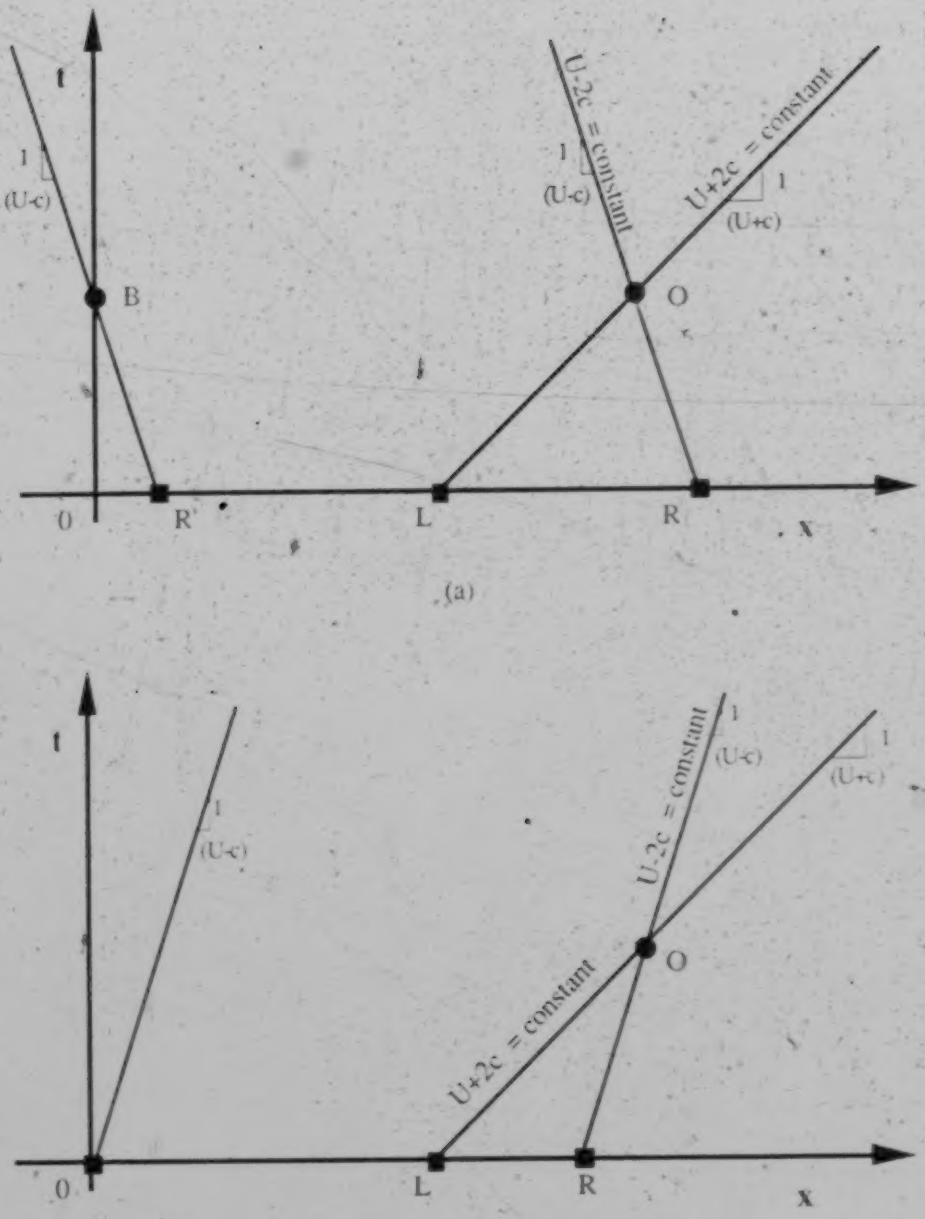
Figure 2.2 Forces acting on the one-dimensional control volume: (a) profile; (b) plan view (resultants).



(a)

(b)

Figure 2.3 One-dimensional control volume for a stationary hydraulic jump.


t
1
(U-c)
B
0
R'
U-2c = constant
1
(U-c)
U+2c = constant
1
(U+c)
O
L
R
x
t
1
(U-c)
0
U-2c = constant
1
(U-c)
1
(U+c)
O
U+2c = constant
L
R
x

(a)

(b)

Figure 2.4 Method of characteristics for shallow water gravity waves: (a) subcritical flow; (b) supercritical flow.

3. NUMERICAL SOLUTION OF THE EQUATIONS OF ONE-DIMENSIONAL OPEN CHANNEL FLOW

3.1 INTRODUCTION

The concept of a kinematic or a shallow water gravity wave, although illustrative, is not convenient for solving practical open channel flow problems. This is because it is not always known a priori which flow approximation will be appropriate and more that one approximation may be required within the domain of concern (either spatial or temporal). As this is not a desirable or even appropriate approach, an alternative is to seek a numerical solution to the complete St. Venant equations.

The approach for any numerical scheme may be generalized into three steps: discretization ; transformation; and solution (Burnett, 1987). First the problem is discretized by dividing the domain into subregions. The second step is the transformation which is achieved by approximating the governing equations with algebraic equations for each subregion. Finally, the resulting set of algebraic equations is solved using either a direct or indirect matrix solver.

In general, numerical methods may be categorized by the way in which they handle the discretization and transformation of a problem (Burnett, 1987). Those used to solve the unsteady open channel flow equations may be classified into three general groups, although there may be some overlap, as:

a) the method of characteristics;

b) finite difference methods (or FDM); and

c) finite element methods (or FEM).

Although this research concentrates on the finite element method, each of the other two methods will be discussed briefly for comparative purposes. As several of the methods discussed have been introduced through the simple linear convection equation [2.96], it will be used in this discussion, as well.

3.2 THE METHOD OF CHARACTERISTICS

The method of characteristics, as discussed in Chapter 2, is an analytical technique for solving hyperbolic equations. However, the resulting ordinary differential equations may be solved numerically as well. A number of authors (Liggett and Cunge, 1975; Abbott, 1979; Cunge, Holly, and Verwey, 1980) provide detailed descriptions of the theory involved in applying the method of characteristics to the one-dimensional unsteady flow equations both analytically and numerically. Katopodes and Strelkoff (1979) have illustrated the applicability of the method of characteristics to the two-dimensional unsteady flow equations, as well. However, for the two dimensional problem , the resulting system are not ordinary differential equations and therefore cross derivatives must be evaluated.

One disadvantage of the method of characteristics lies in the implicit requirement that flow variables must be continuous. At discontinuities the characteristic equations break down and new (shock) relationships must be introduced. This becomes particularly cumbersome when these shocks interact or are reflected off of boundaries (Liggett and Cunge, 1975).

Although the method of characteristics is indispensable as an analytical tool, providing valuable insight into the physical understanding of unsteady open channel flow, its most common numerical use is in defining boundary conditions for some finite difference methods (Liggett and Cunge, 1975).

3.3 FINITE DIFFERENCE METHODS

3.3.1 Introduction

The finite difference method (FDM) is based on a pointwise approximation of the governing differential equations with a set of difference equations. For example, using the grid shown in Figure 3.1, the following substitutions could be made to determine the spatial and temporal derivatives:

$$\frac{\partial\psi}{\partial x} = \frac{\Psi_{j+1}^{n+1} - \Psi_{j-1}^{n+1}}{2\Delta x} \qquad [3.1]$$

$$\frac{\partial\psi}{\partial t} = \frac{\Psi_{j}^{n+1} - \Psi_{j}^{n}}{\Delta t} \qquad [3.2]$$

where, Ψ represents the vector of nodal values of the solution, ψ.

Different schemes are characterized by the manner in which these derivatives are approximated. The FDM has the advantage of being very easy to interpret on a physical basis, and has been used to develop a wide variety of numerical schemes to solve unsteady flow, particularly in the one-dimensional case. For two-dimensional unsteady flow problems, the irregular and solution dependent two-dimensional boundary is difficult to represent on an even finite difference grid (Lee and Froehlich, 1986). Therefore, more complicated coordinate systems or uneven grid spacings are employed.

Discussions of the various finite difference schemes available is beyond the scope of this study and the reader is referred to a variety of authors (Liggett and Cunge, 1975; Abbott, 1979; Cunge, Holly, and Verwey, 1980) who provide detailed descriptions of the application of the various finite difference schemes available. Only the most pertinent cases are discussed herein.

3.3.2 Upwind Finite Differences

A finite difference equation may be written for the method of characteristics based on the grid shown in Figure 3.2, by assuming that:

$$\frac{\partial \psi}{\partial t} = \frac{\Psi_j^{n+1} - \Psi_j^n}{\Delta t} \qquad [3.3]$$

$$\frac{\partial \psi}{\partial x} = \frac{\Psi_{j-1}^n - \Psi_j^n}{\Delta x} \qquad [3.4]$$

Substituting into the linear convection equation, [2.96], the differential equation becomes a difference equation:

$$\frac{\Psi_j^{n+1} - \Psi_j^n}{\Delta t} + U \left(\frac{\Psi_{j-1}^n - \Psi_j^n}{\Delta x} \right) = 0 \qquad [3.5]$$

The method of characteristics is described as an *explicit* method because, as equation [3.5] shows, the unknown value Ψ_j^{n+1} is determined based entirely on known values from the previous time step (n).

The truncation error in making this finite difference approximation to the differential equation may be determined by considering a Taylor series expansion of the terms. For example,

$$\Psi_j^{n+1} = \Psi_j^n + \Delta t \left(\frac{\partial \psi}{\partial t} \right)_j^n + \frac{\Delta t^2}{2} \left(\frac{\partial^2 \psi}{\partial t^2} \right)_j^n + \ldots \qquad [3.6]$$

therefore,

$$\frac{\Psi_j^{n+1} - \Psi_j^n}{\Delta t} = \left(\frac{\partial \psi}{\partial t} \right)_j^n + \frac{\Delta t}{2} \left(\frac{\partial^2 \psi}{\partial t^2} \right)_j^n \qquad [3.7]$$

Similarly,

$$\Psi^n_{j-1} = \Psi^n_j - \Delta x \left(\frac{\partial\psi}{\partial x}\right)^n_j + \frac{\Delta x^2}{2}\left(\frac{\partial^2\psi}{\partial x^2}\right)^n_j - \dots \qquad [3.8]$$

therefore,

$$\frac{\Psi^n_{j-1} - \Psi^n_j}{\Delta x} = \left(\frac{\partial\psi}{\partial x}\right)^n_j - \frac{\Delta x}{2}\left(\frac{\partial^2\psi}{\partial x^2}\right)^n_j \qquad [3.9]$$

Substituting into the difference equation, [3.5],

$$\left(\frac{\partial\psi}{\partial t}\right)^n_j + \frac{\Delta t}{2}\left(\frac{\partial^2\psi}{\partial t^2}\right)^n_j + U\left\{\left(\frac{\partial\psi}{\partial x}\right)^n_j - \frac{\Delta x}{2}\left(\frac{\partial^2\psi}{\partial x^2}\right)^n_j\right\} = 0 \qquad [3.10]$$

and using,

$$\frac{\partial}{\partial t}\left(\frac{\partial\psi}{\partial t}\right) = \frac{\partial}{\partial t}\left(-U\frac{\partial\psi}{\partial x}\right) \qquad [3.11]$$

it becomes:

$$\left(\frac{\partial\psi}{\partial t}\right)^n_j + U\left(\frac{\partial\psi}{\partial x}\right)^n_j + \frac{\Delta t}{2}\frac{\partial}{\partial t}\left\{\left(\frac{\partial\psi}{\partial t}\right)^n_j + \frac{\Delta x}{U\,\Delta t}\left(U\frac{\partial\psi}{\partial x}\right)^n_j\right\} = 0 \qquad [3.12]$$

It can be seen from Figure 3.2 that if,

$$\frac{U\Delta t}{\Delta x} = 1 \qquad [3.13]$$

$$\left(\frac{\partial\psi}{\partial t}\right)^n_j + \frac{\Delta x}{U\,\Delta t}\left(U\frac{\partial\psi}{\partial x}\right)^n_j = 0 \qquad [3.14]$$

so that,

$$\left(\frac{\partial\psi}{\partial t}\right)^n_j + U\left(\frac{\partial\psi}{\partial x}\right)^n_j = 0 \qquad [3.15]$$

and since it can be shown that higher order terms in [3.6] and [3.8] will cancel out similarly, the solution at Ψ^n_{j-1} will be transferred exactly to Ψ^{n+1}_j without truncation error.

Therefore, the method of characteristics provides an exact solution when equation [3.13] is satisfied.

To obtain an exact solution, the grid spacing must depend upon the velocity, U. However, because the velocity is not necessarily constant in space or time, uneven grid intervals result. Consequently, interpolation between nodal values must be used to determine a solution at a prescribed location and time, negating the advantage of an exact solution to some extent. Furthermore, when U is one of the dependent variables this grid tends to become very cumbersome. An alternative is to use a fixed grid and interpolate the dependent variables when solving the characteristic equations.

As illustrated in Figure 3.3, when:

$$\frac{U\Delta t}{\Delta x} > 1 \qquad [3.16]$$

the node of interest (j, n+1), is beyond the region of influence of the previous point (j-1, n) and the information required to obtain a solution is not available. Equation [3.16], therefore, defines a necessary criteria for stability of the method of characteristics (Cunge, Holly, and Verwey, 1980), that is:

$$C = \frac{U\Delta t}{\Delta x} \leq 1 \qquad [3.17]$$

where C is known as the Courant number. The stability criteria is known as the unit Courant- Fredrichs-Lewy (CFL) condition, (Richtmyer and Morton, 1967) and is typical of explicit methods.

3.3.3 The Box Finite Difference Scheme

If one had access to information about all points in the domain during the computations, rather than just those at previous time steps, then the unit CFL condition could potentially be relaxed. This is, in fact, the basis for *implicit* methods (Liggett and Cunge, 1975). Of the many implicit finite difference schemes that have been proposed and used, the four-point schemes are perhaps the most common. As illustrated in Figure 3.4, these schemes are based upon differences defined between adjacent points in the finite difference grid. For example, using the linear convection equation, [2.96], the temporal and spatial derivatives may be defined as:

$$\frac{\partial \psi}{\partial t} = \frac{\Psi_j^{n+1} - \Psi_j^n}{2\Delta t} + \frac{\Psi_{j+1}^{n+1} - \Psi_{j+1}^n}{2\Delta t} \qquad [3.18]$$

$$\frac{\partial \psi}{\partial x} = \theta\left(\frac{\Psi_{j+1}^{n+1} - \Psi_j^{n+1}}{\Delta x}\right) + (1-\theta)\left(\frac{\Psi_{j+1}^n - \Psi_j^n}{\Delta x}\right) \qquad [3.19]$$

respectively, where θ represents the implicitness (i.e. θ = 1 would be a fully implicit case). When the value of θ is set equal to 0.5, this scheme is second order accurate and is equivalent to centered differences in time and space about the center of the box (Amien and Fang, 1975). It is therefore known as the 'box' scheme. Substituting these differences into equation [2.92],

$$\Psi_j^{n+1} - \Psi_j^n + \Psi_{j+1}^{n+1} - \Psi_{j+1}^n + 2\theta\frac{U\Delta t}{\Delta x}\left(\Psi_{j+1}^{n+1} - \Psi_j^{n+1}\right)$$

$$+ 2(1-\theta)\frac{U\Delta t}{\Delta x}\left(\Psi_{j+1}^n - \Psi_j^n\right) = 0 \qquad [3.20]$$

and rearranging gives:

$$\Psi_j^{n+1} + \Psi_{j+1}^{n+1} + 2\theta\frac{U\Delta t}{\Delta x}\left(\Psi_{j+1}^{n+1} - \Psi_j^{n+1}\right)$$

$$= \Psi_j^n + \Psi_{j+1}^n + 2(1-\theta)\frac{U\Delta t}{\Delta x}\left(\Psi_{j+1}^n - \Psi_j^n\right) \qquad [3.21]$$

Note that there are two unknown values, Ψ_j^{n+1} and Ψ_{j+1}^{n+1} at the new time step (n+1) in equation [3.28].

Four-point schemes have been applied with relative success to the one- and two-dimensional unsteady open channel flow equations for quite a number of years. Priessmann (1961) applied the box scheme to a linearized form of the St. Venant equations, finding it to provide both consistent and stable second order accurate solutions. Amien (1968; Amien and Fang, 1970) applied the box scheme to this non-linear system using a Newton-Raphson iterative solution. Fread (1988) has taken this last approach one step further by introducing a variable θ (ranging from 0.5 to 1.0) into the National Weather Service (NWS) DAMBRK model. Because of the general acceptance of the DAMBRK program both in the United States and here in Canada, it was felt that this scheme should be compared to the finite element models studied in this research project. Although the model incorporates a variable implicitness, for convenience it is denoted in this thesis as the 'Box Finite Difference' (BFD) scheme.

Applying the BFD scheme to the one-dimensional continuity equation,

$$\frac{\partial H}{\partial t} + U\frac{\partial H}{\partial x} + H\frac{\partial U}{\partial x} = \frac{\partial H}{\partial t} + \frac{\partial UH}{\partial x} = 0 \qquad [2.55]$$

the following finite difference equation is obtained;

$$H_j^{n+1} + H_{j+1}^{n+1} + \frac{2\theta\Delta t}{\Delta x}\left(U_{j+1}^{n+1}H_{j+1}^{n+1} - U_j^{n+1}H_j^{n+1}\right)$$
$$= H_j^n + H_{j+1}^n + \frac{2\left(1-\theta\right)\Delta t}{\Delta x}\left(U_{j+1}^n H_{j+1}^n - H_j^n H_j^n\right) \qquad [3.22]$$

Similarly, for the longitudinal momentum equation

$$\frac{\partial U}{\partial t} + U\frac{\partial U}{\partial x} + g\frac{\partial H}{\partial x} = g\cdot(S_o - S_f) \qquad [2.56]$$

the following finite difference equation is obtained

$$U_j^{n+1} + U_{j+1}^{n+1} + \frac{\Delta t}{\Delta x}\left\{\theta\left[U_{j+1}^{n+1} + U_j^{n+1}\right] + \left(1-\theta\right)\left[U_{j+1}^n + U_j^n\right]\right\}$$

$$\begin{aligned}
&\left\{ \theta \left[U_{j+1}^{n+1} - U_j^{n+1}\right] + (1-\theta)\left[U_{j+1}^{n} - U_j^{n}\right]\right\} + \\
&g\,\theta\,\Delta t \left\{\frac{2}{\Delta x}\left[H_{j+1}^{n+1} - H_j^{n+1}\right] + \left[(S_f)_{j+1}^{n+1} + (S_f)_j^{n+1}\right]\right\} \\
&= U_j^n + U_{j+1}^n + \frac{2\,\Delta t}{\Delta x}\left[(z_o)_{j+1}^n - (z_o)_j^n\right] \\
&+ g(1-\theta)\,\Delta t \left\{\frac{2}{\Delta x}\left[H_{j+1}^{n} - H_j^{n}\right] + \left[(S_f)_{j+1}^{n} + (S_f)_j^{n}\right]\right\}
\end{aligned} \qquad [3.23]$$

As stated above, this scheme is second order accurate when θ is set to 0.5. However, it is only first order accurate when θ is set to 1.0 (Cunge, 1976). Ponce *et al* (1978) determined that a value of $\theta \leq 0.5$ always leads to numerical amplification, resulting in solution instabilities. They also found that $\theta = 1.0$ always caused numerical attenuation, guaranteeing stability. Several authors (Amien and Chu, 1975; Ponce *et al* ,1978, 1979) have reported that a value of $\theta = 0.5$ is associated with numerical amplification for some problems leading to unstable solutions. Ponce *et al* (1979) recommend $0.6 \leq \theta \leq 0.8$ to achieve a balance between accuracy and stability while Fread (1988) recommends a value of $\theta = 0.6$ as optimum. Other, 'ad-hoc' remidies have used to achieve the same objectives.

These equations, in combination with appropriate initial and boundary conditions, provide a set of 2N non-linear discrete equations, where N is defined as the number of nodes within the domain. Therefore, an iterative solution method is required. The complete set of finite difference equations may be abbreviated as,

$$\{f_k(\Phi)\} = \{0\} \qquad [3.24]$$

where Φ represents the vector of unknowns, that is the nodal values {U, H} and,

$$k = 1 \rightarrow 2N \qquad [3.25]$$

The iterative solution is achieved by first guessing the unknown values at the new time step, Φ^{n+1}, based on the solution at the previous time step (n). Upon substitution of these guesses, the right hand side of equation [3.24] may be evaluated. Unless the guesses are correct, the computed values for the right hand side will not be zeros. In this case these nonzero values are termed the *residual*, $\{\mathbf{R}\}$, such that:

$$\left\{ f_k \left(\Phi^{n+1} \right) \right\} = \{\mathbf{R}\} \neq \{0\} \qquad [3.26]$$

The corrections, $\delta\Phi$, used to obtain the next guess are determined by solving the set of equations based on,

$$[\mathbf{J}]\{\delta\Phi\} = \{\mathbf{R}\} \qquad [3.27]$$

where $[\mathbf{J}]$ is known as the 'Jacobian',

$$[\mathbf{J}] = \left[\frac{\partial \{f_k\}}{\partial \{\Phi\}} \right] = \left[\frac{\partial \{\mathbf{R}\}}{\partial \{\Phi\}} \right] \qquad [3.28]$$

and is evaluated numerically as (Amien and Fang, 1970),

$$\frac{\partial \{\mathbf{R}\}}{\partial \{\Phi\}} = \frac{\{\mathbf{R}(\Phi)\} - \{\mathbf{R}(\Phi\text{-}\delta)\}}{\delta,.} \qquad [3.29]$$

where, δ is a small perturbation. It is important to note that finite difference schemes such as this require separate algorithms based upon the type of flow regime anticipated (subcritical or supercritical). This is due to the different boundary conditions required. For example, in the case of subcritical flow one boundary condition must be specified at each end of the domain. In the case of supercritical flow, two boundary conditions are specified at the upstream end of the domain and therefore the system of equations may be solved two at a time from upstream to downstream, thus simplifying the computations. Though this implies that subcritical and supercritical flow cannot be handled simultaneously, a variety of approaches have been employed to circumvent this problem. For example, Fread (1988) has overcome this restriction through the introduction of a 'mixed flow algorithm' in the

NWS DAMBRK model. This is achieved by breaking the domain up into subreaches which are entirely subcritical or supercritical flow. Transitions between these subreaches are handled as internal boundaries. These subreaches are first defined based on the initial conditions and are then redefined after each time step. Fread reports that the mixed-flow algorithm increases computational expense by about 20 percent.

Due to the limitations on the types of initial and boundary conditions which may be input to the DAMBRK program, it could not be applied to many of the test problems used in this study. Therefore, a program was written explicitly for use in this investigation. In this program the set of equations described by equation [3.27], were solved using a direct solver employing a double sweep algorithm (Zienkiewicz,1977; Stasa, 1985). Convergence was assessed based on the root-mean square of the vector of corrections compared to a specified tolerance, that is if

$$\frac{\sum\left\{(\delta\Phi)^2\right\}}{\sum\left\{(\Phi)^2\right\}} \leq \text{'tolerance'} \qquad [3.30]$$

then the solution would progress to the next step. This criteria was based on the correction rather than the residual for two reasons. First it was simple to compute within the framework of the program developed and second, the vector of unknowns provided a convenient scale for the corrections. No corresponding scale for the residual was readily available.

A complete listing of the finite difference program is provided in Appendix A.

3.4 FINITE ELEMENT METHODS

3.4.1 Introduction

The underlying consistency and generality of the finite element method provides a number of advantages that are particularly suited to this research. For example, its geometric flexibility leads to efficient definition of the irregular features common to natural channels. Also, non-uniform discretizations are easily implemented so that elements can be concentrated in areas where gradients are large, while fewer elements may be used in areas where the flow is more uniform, allowing for optimization of computational effort. Unlike most finite difference methods, which require separate algorithms for subcritical and supercritical flow, the finite element method handles subcritical and supercritical flow in the same way, therefore it can solve for both flow regimes simultaneously. Also, derivative boundary conditions may be implemented without requiring the use of fictitious nodes. Of particular interest to this investigation is the fact that the finite element method extends readily to higher dimensions. This means that numerical schemes which are found to work well for one-dimensional flow, can be considered and tested for the two-dimensional equations more expediently. Because the finite element method is based on intervals (*elements*) rather than points, solutions are obtained over these elements rather than just at the nodes. By varying the functions used to interpolate the solution over the elements, a variety of numerical schemes may be readily implemented and tested. This will be seen in section 3.4.4.

3.4.1.1 Implementation for the Linear Convection Equation

Implementation of the finite element method involves two steps (Burnett, 1987). First of all a trial or approximate solution must be developed. Then this approximate solution must be optimized in some way so as to minimize the error between the trial solution and the exact solution.

Consider the non-conservation form of the linear convection equation [2.96].

$$\frac{\partial \psi}{\partial t} + U \frac{\partial \psi}{\partial x} = 0 \qquad [2.96]$$

An *interpolation* (or *basis*) function, $\mathbf{f}_i$, is used to approximate the behavior of the governing differential equations over each element. Figure 3.5 illustrates a one-dimensional spatial grid and example (linear) interpolation functions, which are dependent only on x. The solution approximation is therefore,

$$\psi = \tilde{\psi} = \sum_{j=1}^{N} \mathbf{f}_j \Psi_j \qquad [3.31]$$

where the trial function $\tilde{\Psi}$ (comprised of the set of interpolation functions) represents the approximation and N is the number of nodes in the domain. The variable Ψ represents the vector of nodal values of ψ, and is therefore independent of x. If the exact solution, ψ, is approximated then equation [2.96] will not necessarily be satisfied. That is,

$$\left\{ \frac{\partial \tilde{\psi}}{\partial t} + U \frac{\partial \tilde{\psi}}{\partial x} \right\} = \{\mathbf{R}\} \neq \{0\} \qquad [3.32]$$

where $\{\mathbf{R}\}$ is the *residual*.

Various techniques may be used to optimize this approximate solution. One way in which to optimize is to use the 'variational' approach, in which some physical quantity, such as energy, is minimized (Burnett, 1987). Unfortunately, this method is limited to problems for which a variational statement may be defined (Lee and Froehlich, 1985). As this is only possible for very restrictive cases of open channel flow, this method was not used in this study. Another way in which to achieve this optimization is by using the method of weighted residuals, in which different weighted averages of the residual, $\{\mathbf{R}\}$, are forced to zero in some way (Burnett, 1987). This includes a subset of optimization techniques such as the point collocation method, the subdomain method and the Galerkin method.

In this study, the element equations were derived using the Galerkin weighted residual method in which a weighted average of the residual is formed and then set equal to zero. This requires the selection of "N" linearly independent *test* (or *weighting*) functions (Lee and Froehlich, 1985), $\mathbf{v}_i$, such that,

$$\int_0^L \mathbf{v}_i \left\{ \mathbf{R} \right\} dx = 0 \qquad [3.33]$$

where L is the length of the domain. These test functions also depend on x and in the original Galerkin approach were taken to be the same as the interpolation functions. This is now known as the Bubnov-Galerkin method, which will be discussed further in section 3.4.2.

Equation, [3.33] is equivalent to:

$$\int_0^L \mathbf{v}_i \left\{ \frac{\partial \tilde{\psi}}{\partial t} + U \frac{\partial \tilde{\psi}}{\partial x} \right\} dx = 0 \qquad [3.34]$$

Equation [3.34] is termed the *weak* statement. Substitution of the interpolation functions from equation [3.31] yields,

$$\int_0^L \mathbf{v}_i \left\{ \frac{\partial \left(\mathbf{f}_j \Psi_j \right)}{\partial t} + U \frac{\partial \left(\mathbf{f}_j \Psi_j \right)}{\partial x} \right\} dx = 0 \qquad [3.35]$$

and because the interpolation functions, $\mathbf{f}_j$, are constant with respect to time and the nodal values Ψ_j do not depend on x, this may be written as,

$$\int_0^L \mathbf{v}_i \left\{ \mathbf{f}_j \frac{\partial \Psi_j}{\partial t} + U\ \Psi_j \frac{\partial \mathbf{f}_j}{\partial x} \right\} dx = 0 \qquad [3.36]$$

or,

$$\int_0^L \left\{ \left(\mathbf{v}_i\, \mathbf{f}_j \frac{\partial \Psi_j}{\partial t} \right) + \left(U\ \Psi_j\, \mathbf{v}_i \frac{\partial \mathbf{f}_j}{\partial x} \right) \right\} dx = 0 \qquad [3.37]$$

This may also be written in matrix notation as:

$$[\mathbf{S}] \left\{ \frac{\partial \Psi}{\partial t} \right\} + [\mathbf{K}] \{\Psi\} = \{0\} \qquad [3.38]$$

where,

$$S_{ij} = \int_0^L \mathbf{v}_i\, \mathbf{f}_j\, dx \qquad [3.39]$$

$$K_{ij} = \int_0^L U\ \mathbf{v}_i \frac{\partial \mathbf{f}_j}{\partial x}\, dx \qquad [3.40]$$

In the finite element method this is actually done on an element level, where equations [3.39] and [3.40] are written for each element as:

$$S_{ij} = \int_{-1}^1 \mathbf{v}_i\, \mathbf{f}_j \frac{dx}{dr}\, dr \qquad [3.41]$$

and,

$$K_{ij} = \int_{-1}^1 \left(U\ \mathbf{v}_i \frac{d\mathbf{f}_j}{dr} \frac{dr}{dx} \right) \frac{dx}{dr}\, dr \qquad [3.42]$$

for,

$$i\,,j = 1, 2, ..., M \qquad [3.43]$$

where r is the local longitudinal coordinate at the element level, ranging from -1 to 1, and M is the number of interpolation functions per element. The local coordinate system is illustrated in Figure 3.6, where the linear trial functions shown are defined by:

$$f_1(r) = \frac{1}{2}(1 - r) \qquad [3.44]$$

$$f_2(r) = \frac{1}{2}(1 + r) \qquad [3.45]$$

Equation [3.37] represents the semi-discrete form of the problem, which is next discretized in time. In this study, this was done using a finite difference approximation such that:

$$\left[\ [\mathbf{S}] + \theta\,\Delta t\,[\mathbf{K}]\ \right]\ \{\Psi^{n+1}\} = \left[\ [\mathbf{S}] + (1-\theta)\,\Delta t\,[\mathbf{K}]\ \right]\ \{\Psi^{n}\} \qquad [3.46]$$

where, again, θ is used to define the level of implicitness. This, combined with appropriate initial and boundary conditions, provides a system of algebraic equations which may be solved using the same direct solver described in section 3.3.3.

3.4.1.2 Implementation for the Unsteady St. Venant Equations

The conservation form of the St. Venant equations in terms of area, A, and discharge, Q, are:

$$\frac{\partial A}{\partial t} + \frac{\partial Q}{\partial x} = 0 \qquad [2.6]$$

$$\frac{\partial Q}{\partial t} + \frac{\partial QU}{\partial x} + \frac{\partial}{\partial x}\left(\frac{gAH}{2}\right) - \frac{gAH}{2B}\frac{dB}{dx} = gA\,(S_o - S_f) \qquad [2.46]$$

Equation [3.46] may be applied to this non-linear system of equations except that here, [**K**] depends upon the solution vector, Φ,

$$\left[[\mathbf{S}] + \theta\,\Delta t\left[\mathbf{K}\left(\Phi^{n+1}\right)\right] \right] \left\{\Phi^{n+1}\right\} = \left[[\mathbf{S}] + (1-\theta)\,\Delta t\left[\mathbf{K}\left(\Phi^{n}\right)\right] \right] \left\{\Phi^{n}\right\} \qquad [3.47]$$

therefore, an iterative solution method is required. As in the case of the implicit finite difference solution of this non-linear system, a Newton-Raphson iteration is employed. However, for the finite element method the Jacobian:

$$[\mathbf{J}] = \left[\frac{\partial\{\mathbf{R}\}}{\partial\{\Phi\}}\right] \qquad [3.28]$$

is determined analytically, rather than numerically. To simplify the calculation an approximation to the Jacobian is often used, for example:

$$[\mathbf{J}] \equiv \left[\frac{\partial\{\mathbf{R}\}}{\partial\{\Phi\}}\right] = \left[[\mathbf{S}] + \theta\,\Delta t\,[\mathbf{K}] \right] \qquad [3.48]$$

This has the advantage of requiring no additional computations per iteration, although as a result the solution may converge too slowly (or not at all). For the purpose of this study, therefore, this approximation was compared against other approximations to the Jacobian for selected methods. Details of these are provided in the ensuing discussion. Convergence was assessed based on the same criteria as used in the BFD program (equation [3.30]).

The contents of the matrix [**S**] remain the same as that described by equation [3.41], except that in this case more than one equation is involved. They may be interwoven to define,

$$\left[S_{ij}\right] = \begin{bmatrix} \int_{-1}^{1} v_i \, f_j \frac{dx}{dr} \, dr & 0 \\ 0 & \int_{-1}^{1} v_i \, f_j \frac{dx}{dr} \, dr \end{bmatrix} \qquad [3.49]$$

For the conservation form of the equations, the weak statement may be integrated by parts, such that [**K**] would be defined as,

$$\left[K_{ij}\right] = \begin{bmatrix} 0 & \int_{-1}^{1} \left(-\frac{dv_i}{dr} \frac{dr}{dx} f_j \right) \frac{dx}{dr} \, dr \\ \int_{-1}^{1} -\frac{gH}{2} \left(\frac{dv_i}{dr} \frac{dr}{dx} f_j \right) \frac{dx}{dr} \, dr & \int_{-1}^{1} -U \left(\frac{dv_i}{dr} \frac{dr}{dx} f_j \right) \frac{dx}{dr} \, dr \end{bmatrix} \qquad [3.50]$$

where friction and slope have been neglected to simplify the discussion. Details of the complete implementation are provided in Appendix B.

The integration by parts introduces additional terms: .

$$\left\{F_{ij}\right\} = \begin{Bmatrix} v_i \left(Q\right)\Big|_0^L \\ v_i \left(UQ + \frac{gH}{2} A\right)\Big|_0^L \end{Bmatrix} \qquad [3.51]$$

such that equation [3.47] would be expanded to:

$$\left[\, [\mathbf{S}] + \theta\,\Delta t \left[\mathbf{K}\left(\Phi^{n+1}\right)\right] \,\right] \left\{\Phi^{n+1}\right\} + \theta\,\Delta t \left\{\mathbf{F}^{n+1}\right\} =$$

$$\left[\, [\mathbf{S}] + (1-\theta)\,\Delta t \left[\mathbf{K}\left(\Phi^{n}\right)\right] \,\right] \left\{\Phi^{n}\right\} + (1-\theta)\,\Delta t \left\{\mathbf{F}^{n}\right\} \qquad [3.52]$$

where the terms in {**F**} represent a flux across the boundaries and are known as *natural* boundary conditions. Specified boundary conditions, therefore, may be implemented directly as contributions to the "right hand side" of the algebraic system. If the dependent variable is unknown, then equation [3.50] defines coefficients contributing to the [**K**] matrix. Natural boundary conditions are particularly valuable when derivative boundary conditions must be implemented (Lee and Froehlich, 1986). In addition, these natural boundary conditions provide compatibility relationships at the nodes facilitating the application of this model to the solution of network problems. For example, at junctions the compatibility relationships defined by equation [3.51] require conservation of mass and longitudinal momentum.

This implementation has illustrated two advantages of using the conservation form of the equations. First, the continuity equation is linear, leaving only one non-linear equation in the system. Second the equations may be integrated by parts, providing a convenient mechanism for implementing boundary conditions. Further details of the implementation for the conservation, non-conservation [2.50] and symmetric [2.60] forms of the equations are provided in Appendix B. A complete listing of the finite element code is provided in Appendix C.

It is the choice of trial and test functions which characterize the various finite element methods used in this study. As will be seen in the ensuing discussion, the convection matrix also plays a vital role in the definition of the various finite element schemes. Consequently, the non-conservation form of the equations is more suitable for describing the methods studied.

3.4.2 The Bubnov-Galerkin Method

The simplest implementation of the finite element method is the Bubnov-Galerkin (BG) approach. As discussed above, in this method the test functions are simply set equal to the basis functions. This is equivalent to centered differences (about each node) in the finite difference method, that is,

$$\frac{\partial \phi}{\partial x} = \theta \left(\frac{\Phi_{j-1}^{n+1} - \Phi_{j+1}^{n+1}}{2\Delta x} \right) + (1-\theta) \left(\frac{\Phi_{j-1}^{n} - \Phi_{j+1}^{n}}{2\Delta x} \right) \qquad [3.53]$$

and,

$$\frac{\partial \phi}{\partial t} = \frac{\Phi^{n+1} - \Phi^{n}}{\Delta t} \qquad [3.54]$$

where,

$$\Phi = \frac{\Phi_{j-1} + 4\Phi_{j} + \Phi_{j+1}}{6} \qquad [3.55]$$

Therefore the implementation of the Bubnov-Galerkin method for the linear convection equation may be written in finite difference form as,

$$\frac{\Phi_{j-1}^{n+1} + 4\Phi_{j}^{n+1} + \Phi_{j+1}^{n+1}}{6} + \theta \left\{ U\Delta t \frac{\Phi_{j-1}^{n+1} - \Phi_{j+1}^{n+1}}{2\Delta x} \right\}$$

$$= \frac{\Phi_{j-1}^{n} + 4\Phi_{j}^{n} + \Phi_{j+1}^{n}}{6} - (1-\theta) \left\{ U\Delta t \frac{\Phi_{j-1}^{n} - \Phi_{j+1}^{n}}{2\Delta x} \right\} \qquad [3.56]$$

The Bubnov-Galerkin formulation extends readily to one-dimensional and two-dimensional systems. However, although this method is useful for modeling relatively flat waves, it performs poorly in the vicinity of large gradients in the solution. Instabilities (spurious oscillations) result and the solution deteriorates rapidly. One way to avoid such oscillations is to refine the discretization (Brooks et al, 1982). Despite these difficulties, the BG method is included in this investigation, both to illustrate its limitations and for comparative purposes.

The complete analytical Jacobian defined by equation [3.28] was employed for this study rather than the approximation defined by equation [3.48], as initial tests indicated that the extra computational effort was offset by faster convergence of the iterative solution. Details of the implementation of the BG scheme are provided in Appendix B.

3.4.3 The Taylor-Galerkin Method

The Taylor-Galerkin (TG) scheme is an explicit formulation ($\theta = 0.0$), in which the unknown variable is written in terms of a Taylor-series expansion in time. The governing equations are then used to devise a time independent formulation, therefore, this method is analogous to the familiar Lax-Wendroff finite difference scheme (Richtmyer and Morton, 1967). In the finite element approach this exploitation of the time-series is used to introduce numerical diffusion (Selmin *et al*, 1985).

The Taylor-Galerkin scheme may be illustrated using the linear convection [2.96] equation by first writing the Taylor-series in time:

$$\psi^{n+1} = \psi^{n} + \Delta t \left(\frac{\partial \psi}{\partial t}\right)^{n} + \frac{\Delta t^2}{2}\left(\frac{\partial^2 \psi}{\partial t^2}\right)^{n} + O\left(\Delta t^3\right) \qquad [3.57]$$

Next, given that,

$$\left(\frac{\partial \psi}{\partial t}\right)^{n} = -\left(\frac{\partial F}{\partial x}\right)^{n} = -U\left(\frac{\partial \psi}{\partial x}\right)^{n} \qquad [3.58]$$

$$\left(\frac{\partial^2 \psi}{\partial t^2}\right)^{n} = -\frac{\partial}{\partial t}\left(\frac{\partial F}{\partial x}\right)^{n} = -\frac{\partial}{\partial x}\left(\frac{\partial F}{\partial t}\right)^{n} = -\frac{\partial}{\partial x}\left(\frac{\partial F}{\partial \psi}\frac{\partial \psi}{\partial t}\right)^{n}$$

$$= -\frac{\partial}{\partial x}\left(U\frac{\partial \psi}{\partial t}\right)^{n} = \frac{\partial}{\partial x}\left(U^2\frac{\partial \psi}{\partial x}\right)^{n} \qquad [3.59]$$

substitution leads to,

$$\psi^{n+1} = \psi^{n} - \Delta t\, U\left(\frac{\partial \psi}{\partial x}\right)^{n} + \frac{\Delta t^2}{2}U^2\left(\frac{\partial^2 \psi}{\partial x^2}\right)^{n} + O\left(\Delta t^3\right) \qquad [3.60]$$

This implementation of the Taylor-Galerkin scheme is equivalent to a Bubnov-Galerkin formulation of the following equation:

$$\frac{\partial \psi}{\partial t} + U\frac{\partial \psi}{\partial x} - \frac{\Delta t}{2}U^2\frac{\partial^2 \psi}{\partial x^2} = 0 \qquad [3.61]$$

Thus apparent diffusion is introduced in the term,

$$\frac{\Delta t}{2} U^2 \frac{\partial^2 \psi}{\partial x^2} \qquad [3.62]$$

Because this is proportional to U^2, it will always be positive, and will therefore lead to numerical attenuation. It is important to note, however, that this is not an arbitrary component of artificial diffusion. It is derived as a term which increases the accuracy of the time discretization over the finite difference approach described in equation [3.46] (Richtmyer and Morton, 1967; Donea, 1984; Selmin *et al.* 1985).

For the case where a source term is involved,

$$\frac{\partial \psi}{\partial t} + U \frac{\partial \psi}{\partial x} + f\psi = 0 \qquad [3.63]$$

the equivalent Bubnov-Galerkin expression would be:

$$\frac{\partial \psi}{\partial t} + U \frac{\partial \psi}{\partial x} + f\psi - \frac{\Delta t}{2} \left\{ \frac{\partial (f\psi)}{\partial \psi} \left(U \frac{\partial \psi}{\partial x} + f\psi \right) + \frac{\partial}{\partial x} \left[U \left(U \frac{\partial \psi}{\partial x} + f\psi \right) \right] \right\} = 0 \qquad [3.64]$$

The Taylor-Galerkin scheme is readily extendible to systems of hyperbolic equations, for example in the one dimensional case:

$$\frac{\partial \{\Phi\}}{\partial t} + [\mathbf{A}] \frac{\partial \{\Phi\}}{\partial x} = 0 \qquad [3.65]$$

would be modeled as the equivalent Bubnov-Galerkin formulation of:

$$\frac{\partial \{\Phi\}}{\partial t} + [\mathbf{A}] \frac{\partial \{\Phi\}}{\partial x} - \frac{\Delta t}{2} \frac{\partial}{\partial x} \left([\mathbf{A}][\mathbf{A}] \frac{\partial \{\Phi\}}{\partial x} \right) = 0 \qquad [3.66]$$

By breaking [A] into components:

$$[A] = [M][\Lambda][M]^{-1} = \begin{bmatrix} \frac{1}{2c} & -\frac{1}{2c} \\ \frac{U+c}{2c} & \frac{U-c}{2c} \end{bmatrix} \begin{bmatrix} U+c & 0 \\ 0 & U-c \end{bmatrix} \begin{bmatrix} -(U-c) & 1 \\ -(U+c) & 1 \end{bmatrix} \qquad [3.67]$$

where $[\Lambda]$ represents the eigenvalues of $[A]$, such that,

$$[\Lambda] \equiv \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \quad \text{where,} \quad \lambda_i = U \pm c \quad \text{for, } i = 1,2 \qquad [3.68]$$

it can be shown that the numerical diffusion introduced will always be positive. That is, since

$$[A]^2 = [M][\Lambda][M]^{-1}[M][\Lambda][M]^{-1} \qquad [3.69]$$

and,

$$[M]^{-1}[M] = [I] \qquad [3.70]$$

then:

$$[A]^2 = [M][\Lambda][I][\Lambda][M]^{-1} \qquad [3.71]$$

Therefore,

$$[A]^2 = [M][\lambda_i^2][M]^{-1} \qquad [3.72]$$

where λ_i is squared and, therefore, always positive.

Note that in the Taylor-Galerkin scheme the size of the diffusion term is controlled by the time step increment. Consequently, the Taylor-Galerkin scheme requires a variable spatial grid in order to adapt to varying flow conditions (Katopodes and Wu, 1986). In addition, the equivalent system of partial differential equations is more complex. Therefore, in a manner analogous the the Lax-Wendroff scheme, additional boundary conditions are required.

Expanding the problem to include source terms (such as the friction and slope terms),

$$\frac{\partial\{\Phi\}}{\partial t} + [\mathbf{A}]\frac{\partial\{\Phi\}}{\partial x} + \{S\} = \{0\} \qquad [3.73]$$

the Taylor-Galerkin method would be equivalent to a Bubnov-Galerkin formulation of the system:

$$\frac{\partial\{\Phi\}}{\partial t} + \left([\mathbf{A}]\frac{\partial\{\Phi\}}{\partial x} + \{S\}\right)$$
$$- \frac{\Delta t}{2}\left(\frac{\partial\{S\}}{\partial\{\Phi\}}\left([\mathbf{A}]\frac{\partial\{\Phi\}}{\partial x} + \{S\}\right) + \frac{\partial}{\partial x}\left([\mathbf{A}]\left([\mathbf{A}]\frac{\partial\{\Phi\}}{\partial x} + \{S\}\right)\right)\right) = \{0\} \qquad [3.74]$$

Though this scheme is termed explicit, the spatial representation still follows equation [3.55] if a *consistent* formulation is used:

$$\Phi = \frac{\Phi_{j-1} + 4\Phi_j + \Phi_{j+1}}{6} \qquad [3.55]$$

This consistency reduces the stability interval of the Taylor-Galerkin method, such that the Courant number, C, as defined in equation [3.17] is limited by:

$$C = \frac{U\Delta t}{\Delta x} \le \frac{1}{\sqrt{3}} = 0.577 \qquad [3.75]$$

rather than by the unit CFL condition applicable to the analogous Lax-Wendroff finite difference scheme (Selmin *et al*, 1985; Katopodes and Wu, 1986; Lee *et al*, 1987; Baker *et al*, 1987). However, if the mass matrix of the FEM Taylor-Galerkin scheme is lumped, it becomes equivalent to the Lax-Wendroff FDM (Selmin *et al*, 1985; Lee *et al*, 1987).

There have been a number of variations on the Taylor-Galerkin approach. For example, Kawahara, Hirano and Tsubota (1982) applied selective lumping to a two-step variation on the Taylor-Galerkin scheme in modelling both one and two-dimensional hyperbolic equations. While it may be argued that the consistent mass matrix is a serious detriment to an explicit scheme, Donea (1984) has illustrated that its advantages can be exploited through a more efficient two-pass scheme. The second-order scheme presented above, as well as a third order scheme, was presented by Donea (1984) and Selmin, Donea and Quartepelle (1985). The third order scheme was found to display the expanded stability range of the Lax-Wendroff FDM (unit CFL condition). It should be noted that in applying the Taylor-Galerkin method to both linear and non linear hyperbolic equations, they found that oscillations tended to appear in the computed solutions. However, they deemed these oscillations to be within tolerable limits. A similar approach entitled the *Characteristic* -Galerkin scheme was presented by Lee *et al* (1987). Katopodes and Wu (1986) also employed a form of the Taylor-Galerkin method, with consistent first and second derivatives in time, coupled with a lumped mass matrix. They noted that the inability of this scheme to handle multiple shocks was a significant limitation. Peraire, Zienkiewicz and Morgan (1986) illustrated the importance of the proper consideration of source and diffusion terms into the Taylor-Galerkin scheme. Minor oscillations were also evident in their results. Peraire *et al* (1986) presented a two-step variation in which both the one and two step schemes were maintained equivalent, at least for the case of a linear system of equations.

The implementation of the Taylor-Galerkin scheme in this study was limited to a consistent second order formulation (equation [3.82]) as well as a lumped or Lax-Wendroff equivalent approach. Details of the implementation are provided in Appendix B. It should be noted that the non-conservation form of the equations were used in the numerical diffusion terms which Peraire *et al* (1986) compare to the "balancing diffusion" or "streamline diffusion" attributed to Donea (1984). Both Peraire *et al* (1986) and Katopodes and Wu (1986) employ the conservation form of the equation in their application.

3.4.4 The Petrov-Galerkin Method

3.4.4.1 Introduction

The problems associated with the Bubnov-Galerkin method have led many researchers to consider the finite element method inappropriate for unsteady open channel flow problems. An alternative is to use the Petrov-Galerkin method, in which upwind weighted test functions are used to introduce numerical diffusion to damp these spurious oscillations.

The schemes considered in this study employ the following form of the test function,

$$v_i = f_i + g_i \qquad [3.76]$$

in which,

$$g_i = \omega\, W \frac{\Delta x}{2} \frac{df_i}{dx} \qquad [3.77]$$

where, ω represents a diffusion parameter (or *upwinding coefficient*) dependent upon the type of flow situation, while W represents an upwinding coefficient which controls both the amount and the direction of the numerical diffusion.

In the case of the linear convection equation, this Petrov-Galerkin formulation is equivalent to a Bubnov-Galerkin formulation of the equation,

$$\frac{\partial\psi}{\partial t} + U\frac{\partial\psi}{\partial x} - \omega\, W \frac{\Delta x}{2}\frac{\partial}{\partial x}\left(\frac{\partial\psi}{\partial t} + U\frac{\partial\psi}{\partial x}\right) = 0 \qquad [3.78]$$

thus illustrating the introduction of numerical diffusion.

In the case of one-dimensional unsteady open channel flow, the equivalent Bubnov-Galerkin formulation may be defined in the following form,

$$\begin{bmatrix} f_i & 0 \\ 0 & f_i \end{bmatrix} \begin{Bmatrix} C \\ M \end{Bmatrix} + \omega \frac{\Delta x}{2} [\mathbf{W}] \frac{d}{dx} \begin{bmatrix} f_i & 0 \\ 0 & f_i \end{bmatrix} \begin{Bmatrix} C \\ M \end{Bmatrix} = \{0\} \tag{3.79}$$

where, *C* and *M* refer to the continuity and momentum equations, respectively. In the ensuing discussion the term 'upwinding coefficient' will be used to denote the diffusion parameter ω, while [W] will be referred to as the *upwinding matrix*. Introducing names for the components of the upwinding matrix,

$$[\mathbf{W}] \equiv \begin{bmatrix} w_{aa} & w_{aq} \\ w_{qa} & w_{qq} \end{bmatrix} \tag{3.80}$$

equation [3.79] may be expressed as

$$\begin{bmatrix} f_i & 0 \\ 0 & f_i \end{bmatrix} \begin{Bmatrix} C \\ M \end{Bmatrix} + \omega \frac{\Delta x}{2} \begin{bmatrix} \left(w_{aa} \frac{df_i}{dx}\right) & \left(w_{aq} \frac{df_i}{dx}\right) \\ \left(w_{qa} \frac{df_i}{dx}\right) & \left(w_{qq} \frac{df_i}{dx}\right) \end{bmatrix} \begin{Bmatrix} C \\ M \end{Bmatrix} = \{0\} \tag{3.81}$$

or

$$f_i C + \omega \frac{\Delta x}{2} \left(w_{aa} \frac{df_i}{dx}\right) C + \omega \frac{\Delta x}{2} \left(w_{aq} \frac{df_i}{dx}\right) M = 0 \tag{3.82}$$

for the continuity equation, and

$$f_i M + \omega \frac{\Delta x}{2} \left(w_{qa} \frac{df_i}{dx}\right) C + \omega \frac{\Delta x}{2} \left(w_{qq} \frac{df_i}{dx}\right) M = 0 \tag{3.83}$$

for the momentum equation.

The analytical Jacobian defined by equation [3.28] evaluated for the Bubnov-Galerkin scheme, was used for most of the tests conducted with the Petrov-Galerkin schemes. However, some comparisons were conducted introducing the additional terms associated with the upwinding into the Jacobian to again see if the extra computational effort was offset by faster convergence of the iterative solution. Details of the implementation are shown in Appendix B.

3.4.4.2 The Dissipative-Galerkin Method

This is an implicit method, originally introduced by Katopodes (1984) for solving the St. Venant equations. Implementation of this scheme introduces numerical diffusion through an upwinding term defined by (Katopodes, 1984):

$$\left[\mathbf{v}\right] = \left[\mathbf{f}\right] + \varepsilon\left[\left[\mathbf{A}\right]^{T}\frac{d\mathbf{f}}{dx}\right] \quad \text{where,} \quad \varepsilon = \frac{\Delta x}{|U + c|\sqrt{15}} \qquad [3.84]$$

so that, in terms of equation [3.79], implementation of this method requires that:

$$\omega \equiv \frac{2}{\sqrt{15}} \approx 0.50 \qquad [3.85]$$

and

$$\left[\mathbf{W}\right] \equiv \begin{bmatrix} w_{aa} & w_{aq} \\ w_{qa} & w_{qq} \end{bmatrix} \equiv \frac{\left[\mathbf{A}\right]}{|U + c|} = \begin{bmatrix} 0 & \dfrac{1}{|U + c|} \\ \dfrac{(c^2 - U^2)}{|U + c|} & \dfrac{2U}{|U + c|} \end{bmatrix} \qquad [3.86]$$

This upwinding matrix [W] may be generalized by again considering the components of [A], where,

$$\left[\mathbf{A}\right] = \left[\mathbf{M}\right]\left[\Lambda\right]\left[\mathbf{M}\right]^{-1} = \begin{bmatrix} \dfrac{1}{2c} & -\dfrac{1}{2c} \\ \dfrac{U + c}{2c} & \dfrac{U - c}{2c} \end{bmatrix} \begin{bmatrix} U + c & 0 \\ 0 & U - c \end{bmatrix} \begin{bmatrix} -(U - c) & 1 \\ -(U + c) & 1 \end{bmatrix} \qquad [3.67]$$

and,

$$\left[\Lambda\right] \equiv \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \quad \text{where,} \quad \lambda_i = U \pm c \quad \text{for, } i = 1,2 \qquad [3.68]$$

by substituting equation [3.67] into [3.86] to obtain,

$$[\mathbf{W}] = [\mathbf{M}]\left[\frac{\lambda_i}{|\lambda_i|}\right][\mathbf{M}]^{-1} = \begin{bmatrix} \frac{1}{2c} & -\frac{1}{2c} \\ \frac{U+c}{2c} & \frac{U-c}{2c} \end{bmatrix} \begin{bmatrix} \frac{U+c}{|U+c|} & 0 \\ 0 & \frac{U-c}{|U+c|} \end{bmatrix} \begin{bmatrix} -(U-c) & 1 \\ -(U+c) & 1 \end{bmatrix} \qquad [3.87]$$

This method has provided very promising results for a number of one-dimensional tests, including shocks (Katopodes, 1984). However, for some problems, the value of ω had to be increased beyond the value given by equation [3.86] to assure sufficient damping. This method has been also been successfully extended to two-dimensional problems (Katopodes, 1984, 1987; Wang and Adeff, 1987). Details of the implementation of the Dissipative-Galerkin method for the one-dimensional equations are provided in Appendix B.

3.4.4.3 The Characteristic-Dissipative-Galerkin Method

This Petrov-Galerkin scheme was originally introduced by Brooks and Hughes (1982) as the Streamline Upwind Petrov-Galerkin (SU/PG). In this approach, the numerical diffusion was incorporated using an upwinding term which was determined based upon the sign of the flow direction. Although this method was addressed to problems in the area of gas dynamics it may be illustrated as an equivalent Bubnov-Galerkin formulation for the linear convection equation:

$$\frac{\partial \psi}{\partial t} + U\frac{\partial \psi}{\partial x} - \omega\frac{U}{|U|}\frac{\Delta x}{2}\frac{\partial}{\partial x}\left(\frac{\partial \psi}{\partial t} + U\frac{\partial \psi}{\partial x}\right) = 0 \qquad [3.88]$$

Upon initial examination, this appears analogous to the Dissipative Galerkin scheme of Katopodes, described above. In fact, in a subsequent paper (Hughes and Tezduyar, 1984) an extension of the SU/PG scheme to handle systems of hyperbolic equations employed that approach. However, examination of the extension to systems lead their investigations to consider a decoupling of the one-dimensional system, given that a disturbance may be propagated at more than one celerity (Hughes *et al*, 1986). This was seen in the characteristic form of the equations:

$$\frac{D\,(U \pm 2c)}{Dt} = \frac{\partial(U \pm 2c)}{\partial t} + \begin{bmatrix} U+c & 0 \\ 0 & U-c \end{bmatrix} \frac{\partial(U \pm 2c)}{\partial x} = g\,(S_o - S_f) \qquad [2.103]$$

Therefore, implementation of SU/PG for systems of hyperbolic equations, requires the proper generalization of:

$$W = \frac{U}{|U|} \qquad [3.89]$$

which is analogous to:

$$[\mathbf{W}] = \frac{[\mathbf{A}]}{|[\mathbf{A}]|} \qquad [3.90]$$

which becomes,

$$[\mathbf{W}] \equiv \begin{bmatrix} w_{aa} & w_{aq} \\ w_{qa} & w_{qq} \end{bmatrix} \equiv \frac{1}{2c} \begin{bmatrix} \dfrac{(c^2 - U^2)}{|U + c| - |U - c|} & \dfrac{(U + c)}{|U + c|} - \dfrac{(U - c)}{|U - c|} \\ (c^2 - U^2)\left(\dfrac{(U + c)}{|U + c|} - \dfrac{(U - c)}{|U - c|}\right) & \dfrac{(U + c)^2}{|U + c|} - \dfrac{(U - c)^2}{|U - c|} \end{bmatrix} \qquad [3.91]$$

Again, considering the components of [**A**], equation [3.91] generalizes to:

$$[\mathbf{W}] = [\mathbf{M}]\left[\frac{\lambda_i}{|\lambda_i|}\right][\mathbf{M}]^{-1} = \begin{bmatrix} \dfrac{1}{2c} & -\dfrac{1}{2c} \\ \dfrac{U+c}{2c} & \dfrac{U-c}{2c} \end{bmatrix} \begin{bmatrix} \dfrac{U+c}{|U+c|} & 0 \\ 0 & \dfrac{U-c}{|U-c|} \end{bmatrix} \begin{bmatrix} -(U-c) & 1 \\ -(U+c) & 1 \end{bmatrix} \qquad [3.92]$$

which is indeed quite similar to the Dissipative-Galerkin scheme. However, in this case both characteristic velocities are used in the determination of the upwinding matrix. Therefore, this implementation of the SU/PG for the St. Venant equations will be denoted as the "*Characteristic* -Dissipative-Galerkin" (CDG) method.

Hughes, Franca and Mallet (1986) examined the application of this method to symmetric systems of hyperbolic equations, thus assuring orthogonality of the eigenvectors of the upwinding matrix. Generalization to multi-dimensional systems were also investigated in a subsequent paper (Hughes, Mallet and Mizukami, 1986). In the case of open channel flow, extensions to the two-dimensional equations [2.41] to [2.43] could be implemented for a symmetric system of the form:

$$\frac{\partial}{\partial t}\begin{Bmatrix} U \\ V \\ 2c \end{Bmatrix} + \begin{bmatrix} U & 0 & c \\ 0 & U & 0 \\ c & 0 & U \end{bmatrix} \frac{\partial}{\partial x}\begin{Bmatrix} U \\ V \\ 2c \end{Bmatrix} + \begin{bmatrix} V & 0 & 0 \\ 0 & V & c \\ 0 & c & V \end{bmatrix} \frac{\partial}{\partial y}\begin{Bmatrix} U \\ V \\ 2c \end{Bmatrix} = 0 \qquad [3.93]$$

where friction and slope have been neglected to simplify the discussion.

In this study, the CDG method was implemented for both symmetric and non-symmetric forms of the St. Venant equations in an attempt to determine if the cross-upwinding effect was significant enough to require the use of symmetric forms in general shallow flow modeling. As this requires the use of a non-conservation system, comparison of conservative and non-conservative, non-symmetric formulations were also done. Details of the implementation of the CDG method are provided in Appendix B.

3.4.4.4 Comparison of Upwinding Matrices for the Petrov-Galerkin Method

The upwinding matrix for the DG scheme [3.86] may also be written in terms of Froude number,

$$[\mathbf{W}] \equiv \begin{bmatrix} w_{aa} & w_{aq} \\ w_{qa} & w_{qq} \end{bmatrix} = \begin{bmatrix} 0 & \frac{1}{c}\frac{1}{|\mathbf{F}+1|} \\ c\,\frac{-(\mathbf{F}+1)(\mathbf{F}-1)}{|\mathbf{F}+1|} & \frac{2\,\mathbf{F}}{|\mathbf{F}+1|} \end{bmatrix} \qquad [3.94]$$

similarly for the CDG scheme,

$$[\mathbf{W}] \equiv \begin{bmatrix} w_{aa} & w_{aq} \\ w_{qa} & w_{qq} \end{bmatrix} = \begin{bmatrix} \frac{1}{2}\frac{(1 - \mathbf{F}^2)}{|\mathbf{F}+1| - |\mathbf{F}-1|} & \frac{1}{2c}\left(\frac{(\mathbf{F}+1)}{|\mathbf{F}+1|} - \frac{(\mathbf{F}-1)}{|\mathbf{F}-1|}\right) \\ \frac{c}{2}(1 - \mathbf{F}^2)\left(\frac{(\mathbf{F}+1)}{|\mathbf{F}+1|} - \frac{(\mathbf{F}-1)}{|\mathbf{F}-1|}\right) & \frac{1}{2}\left(\frac{(\mathbf{F}+1)^2}{|\mathbf{F}+1|} - \frac{(\mathbf{F}-1)^2}{|\mathbf{F}-1|}\right) \end{bmatrix} \qquad [3.95]$$

The variations of [**W**] with respect to Froude number for the DG and CDG schemes are shown in Figures 3.7 and 3.8 , respectively, illustrating the fundamental differences which distinguish the two schemes and the fact that they are identical at **F** = 0 and 1. In addition, as the expanded Jacobian (discussed in section 3.4.4.1) was evaluated by neglecting the dependence of the upwinding matrix on the solution, it corresponds to the complete analytical Jacobian for the CDG scheme for supercritical flow, where [**W**] is constant with respect to **F**.

An interesting aspect of the CDG scheme becomes apparent when the symmetric formulation is introduced. In this case, the upwinding matrix simplifies to:

$$[\mathbf{W}] = \frac{1}{2}\begin{bmatrix} \left(\frac{U+c}{|U+c|} + \frac{U-c}{|U-c|}\right) & \left(\frac{U+c}{|U+c|} - \frac{U-c}{|U-c|}\right) \\ \left(\frac{U+c}{|U+c|} - \frac{U-c}{|U-c|}\right) & \left(\frac{U+c}{|U+c|} + \frac{U-c}{|U-c|}\right) \end{bmatrix}, \qquad [3.96]$$

which, for the case of U > 0, becomes:

$$[\mathbf{W}] = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \qquad [3.97]$$

for subcritical flow, and

$$[\mathbf{W}] = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad [3.98]$$

for the supercritical case. From these, equations [3.82] and [3.83] simplify to:

$$\left(\mathbf{f}_i\, C + \omega\frac{\Delta x}{2}\frac{d\mathbf{f}_i}{dx} M = 0\right) \text{ and, } \left(\mathbf{f}_i\, M + \omega\frac{\Delta x}{2}\frac{d\mathbf{f}_i}{dx} C = 0\right) \qquad [3.99]$$

for subcritical flow, and

$$\left(f_i\, C + \omega \frac{\Delta x}{2} \frac{df_i}{dx}\, C = 0\right) \text{ and, } \left(f_i\, M + \omega \frac{\Delta x}{2} \frac{df_i}{dx}\, M = 0\right) \qquad [3.100]$$

for supercritical flow.

3.4.4.5 Choice of Upwinding Parameter for the Petrov-Galerkin Method

The value of the upwinding coefficient ω for the SU/PG scheme has been the subject of some investigation. Brooks and Hughes (1978, 1980, 1982) report that for the steady one-dimensional convection diffusion equation, a value of $\omega = 1.0$ is optimum in that it leads to exact solutions at the nodes. For the transient case an optimum can be based on maximizing accuracy or minimizing phase error. For the one-dimensional convection equation Raymond and Garder (1976) showed that a value of $\omega = 2 / \sqrt{15}$ will minimize phase error (Brooks and Hughes, 1982).

Katopodes (1984) examined several values of ω for the Dissipative-Galerkin scheme applied to the open channel flow equations, including $\omega = 2 / \sqrt{15}$, and $\omega = \Delta t\, |U + c| / \Delta x$. He concluded that numerical experimentation would be required to quantify an optimum value. Froehlich (1985) illustrated the effect of varying the upwinding coefficient on the phase and amplitude accuracy of this scheme for the linear equations and demonstrated a great sensitivity to Courant number at the shorter wavelengths. He also suggested that the effects of friction should be considered in the selection of the upwinding coefficient.

Brooks and Hughes (1982) consider the structure of the upwinding matrix much more important to the phase and amplitude accuracy of a particular numerical scheme than the exact value of ω. Therefore, in the interest of minimizing the number of model parameters used in this study, the optimization analysis was limited to the consideration of the sensitivity of these Petrov-Galerkin schemes to the upwinding coefficient.

3.4.5 The Least-Squares Method

The Least-Squares method is is an implicit scheme in which the residual is minimized in terms of least-squares (Jiang and Povinelli, 1989) rather than by the Galerkin method described above. Considering the linear convection equation , and discretizing in time:

$$\frac{\psi^{n+1} - \psi^{n}}{\Delta t} + U\left(\frac{\theta\, d\psi^{n+1}}{dx} + \frac{(1-\theta)\, d\psi^{n}}{dx}\right) = 0 \qquad [3.101]$$

where θ is again indicative of the degree of implicitness, a least-squares residual may be defined as:

$$\mathbf{R} = \int_{\Omega}\left(\frac{\psi^{n+1} - \psi^{n}}{\Delta t} + U\left(\frac{\theta\, d\psi^{n+1}}{dx} + \frac{(1-\theta)\, d\psi^{n}}{dx}\right)\right)^{2} dx$$

$$= \int_{\Omega}\left(\psi^{n+1} - \psi^{n} + \Delta t\; U\left(\frac{\theta\, d\psi^{n+1}}{dx} + \frac{(1-\theta)\, d\psi^{n}}{dx}\right)\right)^{2} dx \qquad [3.102]$$

where Ψ^{n} represents the known solution at the previous time step. Equation [3.102] is second-order accurate for θ=0.5, otherwise it is first-order accurate (Carey and Jiang, 1988). Simplifying, this becomes:

$$\mathbf{R} = \int_{\Omega}\left(\psi^{n+1} + \Delta t\, \theta\, U\, \frac{d\psi^{n+1}}{dx} + \mathbf{F}\right)^{2} dx \qquad [3.103]$$

where **F** represents a number incorporating the Ψ^{n} terms. An approximate solution $\widehat{\Psi}$ may be defined such that:

$$\psi^{n+1} = \widehat{\psi} + \varepsilon v \qquad [3.104]$$

where v is a function (dependent on x but not t) which satisfies the boundary conditions and leads to a perturbation on $\widehat{\Psi}$, and ε is some small number. The objective, then, is to solve for $\widehat{\psi}$ such that **R** is a minimum. This may be achieved by first substituting equation [3.104] into [3.103],

$$\mathbf{R} + \delta\mathbf{R} = \int_{\Omega} \left\{ \left(\widehat{\psi} + \varepsilon v\right) + \Delta t\, \theta\, U \frac{d\left(\widehat{\psi} + \varepsilon v\right)}{dx} + \mathbf{F} \right\}^2 dx \qquad [3.105]$$

Expanding and eliminating terms multiplied by ε^2:

$$\mathbf{R} + \delta\mathbf{R} = \int_{\Omega} \left[\left(\widehat{\psi} + \Delta t\, \theta\, U \frac{d\widehat{\psi}}{dx} + \mathbf{F}\right)^2 + 2\varepsilon\, v \left(\widehat{\psi} + \Delta t\, \theta\, U \frac{d\widehat{\psi}}{dx} + \mathbf{F}\right) \right.$$

$$\left. + \, 2\varepsilon\, \Delta t\, \theta\, U \frac{dv}{dx} \left(\widehat{\psi} + \Delta t\, \theta\, U \frac{d\widehat{\psi}}{dx} + \mathbf{F}\right) \right] dx \qquad [3.106]$$

such that,

$$\delta\mathbf{R} = \int_{\Omega} \left[2\varepsilon\, v \left(\widehat{\psi} + \Delta t\, \theta\, U \frac{d\widehat{\psi}}{dx} + \mathbf{F}\right) + 2\varepsilon\, \Delta t\, \theta\, U \frac{dv}{dx} \left(\widehat{\psi} + \Delta t\, \theta\, U \frac{d\widehat{\psi}}{dx} + \mathbf{F}\right) \right] dx$$

$$= \int_{\Omega} \left[2\varepsilon \left(v + \Delta t\, \theta\, U \frac{dv}{dx}\right) \left(\widehat{\psi} + \Delta t\, \theta\, U \frac{d\widehat{\psi}}{dx} + \mathbf{F}\right) \right] dx \qquad [3.107]$$

A necessary condition for **R** to be a minimum, then, would be:

$$\delta\mathbf{R} = \int_{\Omega} \left(v + \Delta t\, \theta\, U \frac{dv}{dx}\right) \left(\widehat{\psi} + \Delta t\, \theta\, U \frac{d\widehat{\psi}}{dx} + \mathbf{F}\right) dx = 0 \qquad [3.108]$$

And since v represents any function this leads to the weak statement:

$$\int_{\Omega} \left[\psi^{n+1} - \psi^{n} + U\, \Delta t \left\{ \theta \frac{d\psi^{n+1}}{dx} + (1-\theta) \frac{d\psi^{n}}{dx} \right\} \right] \left(\mathbf{f}_i + U\, \Delta t\, \theta \frac{d\mathbf{f}_i}{dx} \right) dx = 0 \qquad [3.109]$$

This has the form of a Petrov-Galerkin method in that:

$$\mathbf{v}_i = \mathbf{f}_i + \Delta t\,\theta\,U\frac{d\mathbf{f}_i}{dx} \qquad [3.110]$$

as in equations [3.76] and [3.77], though with two notable exceptions. First the appearance of the term $\mathbf{f}_i$, which occurs only due to the transient nature of the problem. The second contrast is in the coefficient of the derivative in the second term of equation [3.110] where the amount of numerical diffusion added in this scheme depends both on Δt and θ. For example, as the time step increment is reduced, this method approaches a Bubnov-Galerkin formulation.

This dependence of the numerical diffusion on Δt also illustrates one of the similarities between the Least-Squares method and the Taylor-Galerkin method. Another is the necessity to discretize in time first, then in space. The third comparison requires consideration of the source terms. Unlike the Petrov-Galerkin methods discussed above, the source terms are incorporated in the upwinding. Consider again the system:

$$\frac{\partial\{\Phi\}}{\partial t} + [\mathbf{A}]\frac{\partial\{\Phi\}}{\partial x} + \{S\} = \{0\} \qquad [3.73]$$

Implementation of the Least-Squares method for this equations would be equivalent to a Bubnov-Galerkin formulation of the system:

$$\begin{aligned}&\frac{\partial\{\Phi\}}{\partial t} + [\mathbf{A}]\frac{\partial\{\Phi\}}{\partial x} + \{S\}\\ &\quad - \Delta t\,\theta\left(\frac{\partial[\mathbf{A}]}{\partial x} + \frac{\partial\{S\}}{\partial\{\Phi\}}\right)\left(\frac{\partial\{\Phi\}}{\partial t} + [\mathbf{A}]\frac{\partial\{\Phi\}}{\partial x} + \{S\}\right)\\ &\quad - \Delta t\,\theta\,\frac{\partial}{\partial x}\left([\mathbf{A}]^T\left(\frac{\partial\{\Phi\}}{\partial t} + [\mathbf{A}]\frac{\partial\{\Phi\}}{\partial x} + \{S\}\right)\right) = \{0\}\end{aligned} \qquad [3.111]$$

Details of the implementation of the Least-Squares method are provided in Appendix B.

3.4.6 Other Finite Element Methods

Two other finite element methods which were not included in this investigation are the wave equation and mixed order interpolation schemes. In the wave equation method numerical diffusion is introduced through the use of a wave continuity equation (Kinnmark, 1986). Neglecting width variations and the effects of friction, the wave equation method for the St. Venant system has the form,

$$\frac{\partial}{\partial t}\left\{\frac{\partial A}{\partial t}+\frac{\partial Q}{\partial x}\right\} - \frac{\partial}{\partial x}\left\{\frac{\partial Q}{\partial t} + \frac{\partial QU}{\partial x} + \frac{\partial}{\partial x}\left(\frac{gAH}{2}\right)\right\} = \{0\} \qquad [3.112]$$

thus introducing artificial diffusion through the second order space derivative (Kinnmark, 1986). Lynch and Gray (1979) Kinnmark and Gray (1982) and Kinnmark (1986) have all exhibited the ability of this scheme to suppress spurious oscillations when modeling tidal estuaries using implicit finite element algorithms. One disadvantage of this scheme is the requirement for a three time level temporal discretization which has been shown to introduce time oscillations in the velocity solution (Kinnmark, 1986). In addition, it is addressed to typically subcritical flows and is, in fact, somewhat analogous in approach to the application of the symmetric CDG for subcritical flows defined in equation [3.99].

Hood and Taylor (1974), Platzman (1981) and Walters and Carey (1983) are just some of the investigators who have applied mixed order interpolation methods to the shallow flow equations (Lee and Froehlich, 1986). As the name implies, different interpolation functions are taken for the dependent variables making these methods analogous to the staggered finite difference approach (Lee and Froehlich, 1986). Typically, a higher order function is used for the velocity than the depth. Hood and Taylor (1974) attribute the need for mixed order interpolation to a necessity for error consistency (Lee and Froehlich, 1986). The success of both these approaches in suppressing spurious oscillations has also been attributed to their ability to propagate 2Δx waves (Kinnmark, 1986; Lee and Froehlich, 1986).

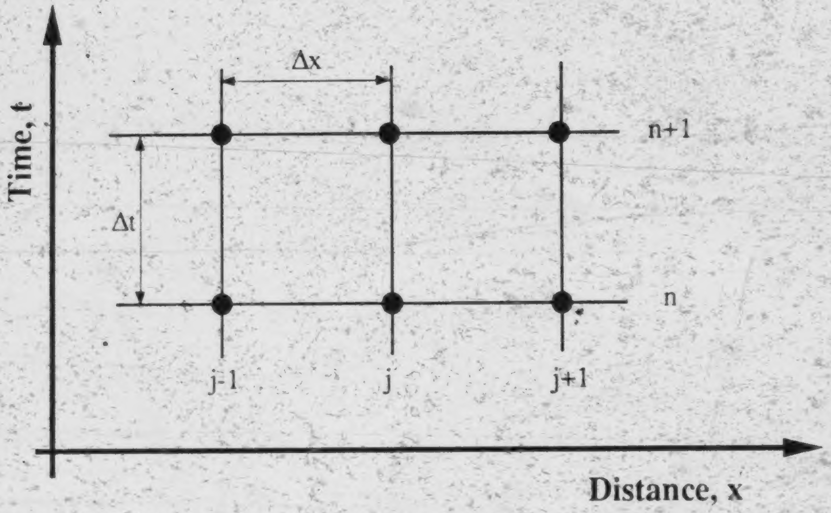

Time, t
Δx
Δt
n+1
n
j-1
j
j+1
Distance, x

Figure 3.1 Definition sketch for the finite difference grid for one-dimensional flow.

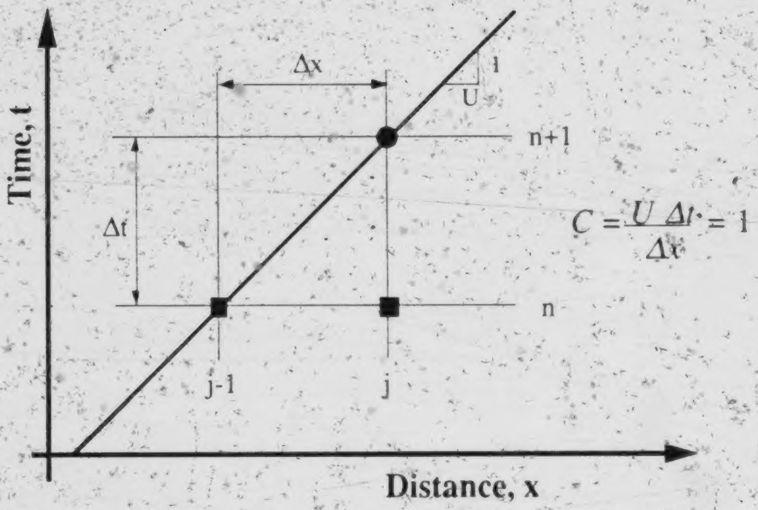

Time, t
Δx
1
U
n+1
Δt
$C = \frac{U\,\Delta t}{\Delta x} = 1$
n
j-1
j
Distance, x

Figure 3.2 The method of characteristics for the one-dimensional convection equation.

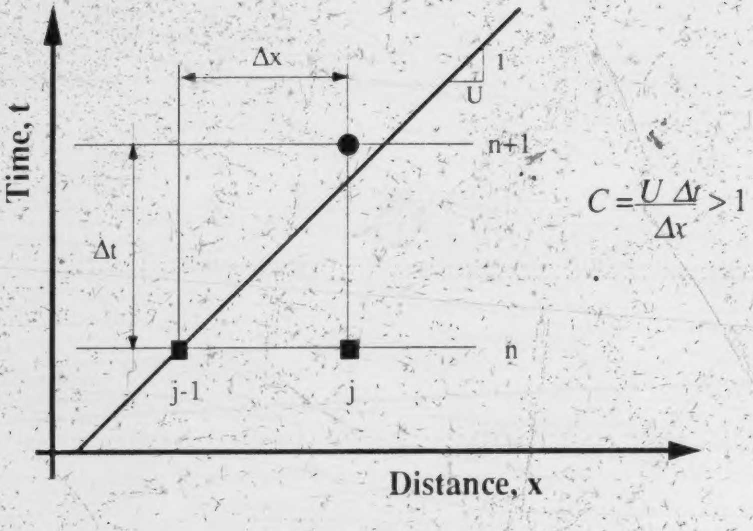

Time, t
Distance, x
Δx
Δt
1
U
n+1
n
j-1
j
$C = \frac{U\,\Delta t}{\Delta x} > 1$

Figure 3.3 Violation of the stability criteron in the method of characteristics.

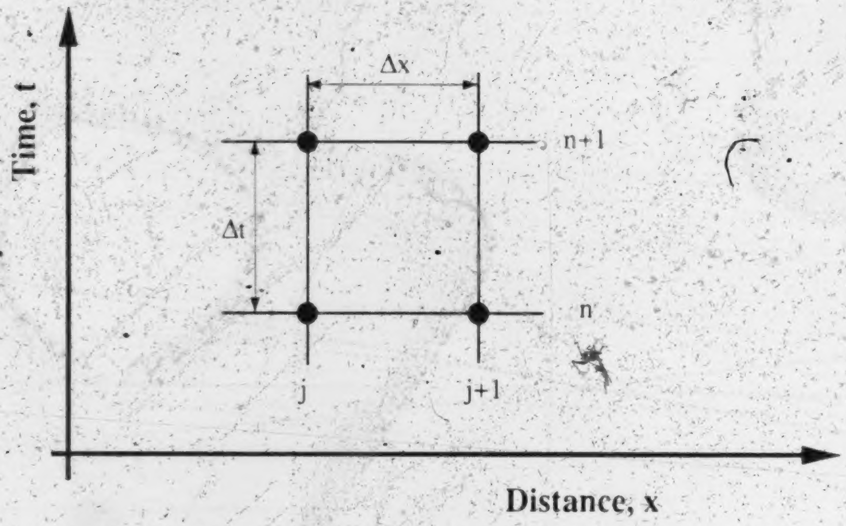

Time, t
Δx
n+1
Δt
n
j
j+1
Distance, x

Figure 3.4 Definition sketch for the finite difference grid for four-point schemes.

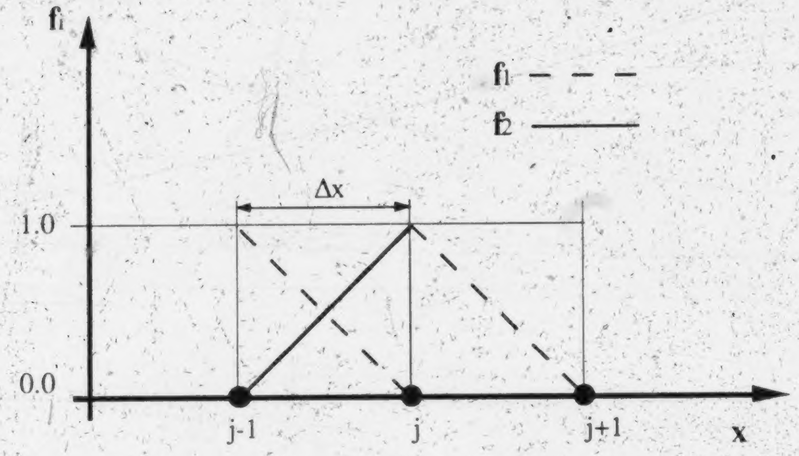

fi
f1
f2
Δx
1.0
0.0
j-1
j
j+1
x

Figure 3.5 Definition sketch for the finite element method for one-dimensional flow.

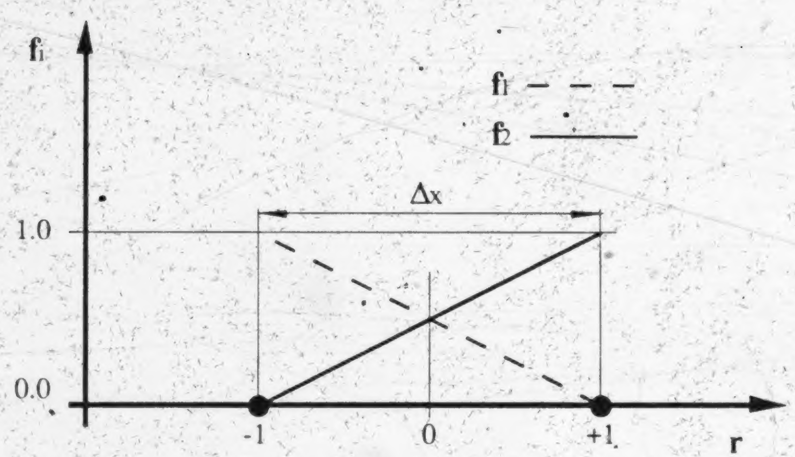

f_i
f_1
f_2
Δx
1.0
0.0
-1
0
+1
r

Figure 3.6 Definition sketch for the local coordinate, r, within an element method for one-dimensional flow.

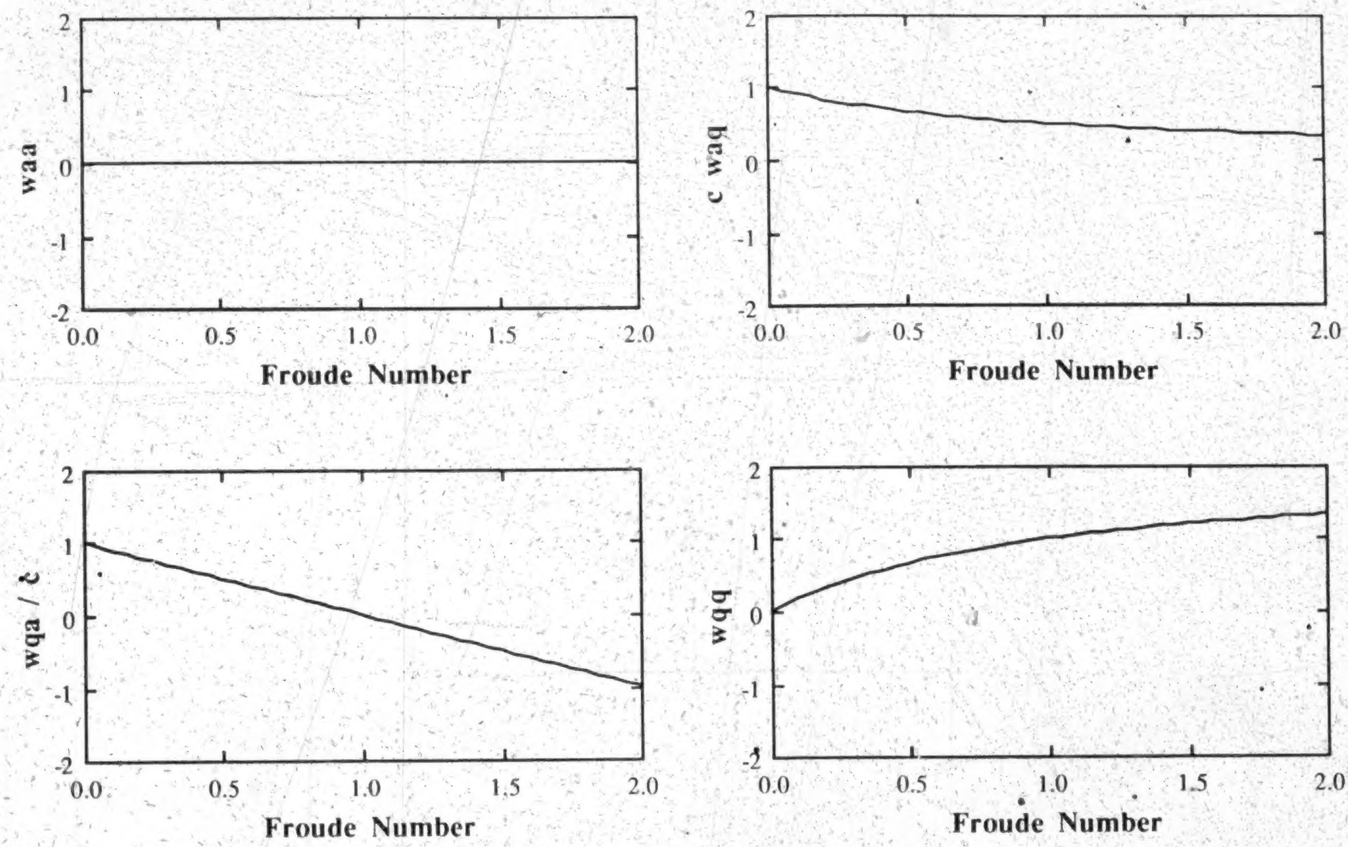


Figure 3.7 Variation of [**W**] with **F** for the DG scheme.

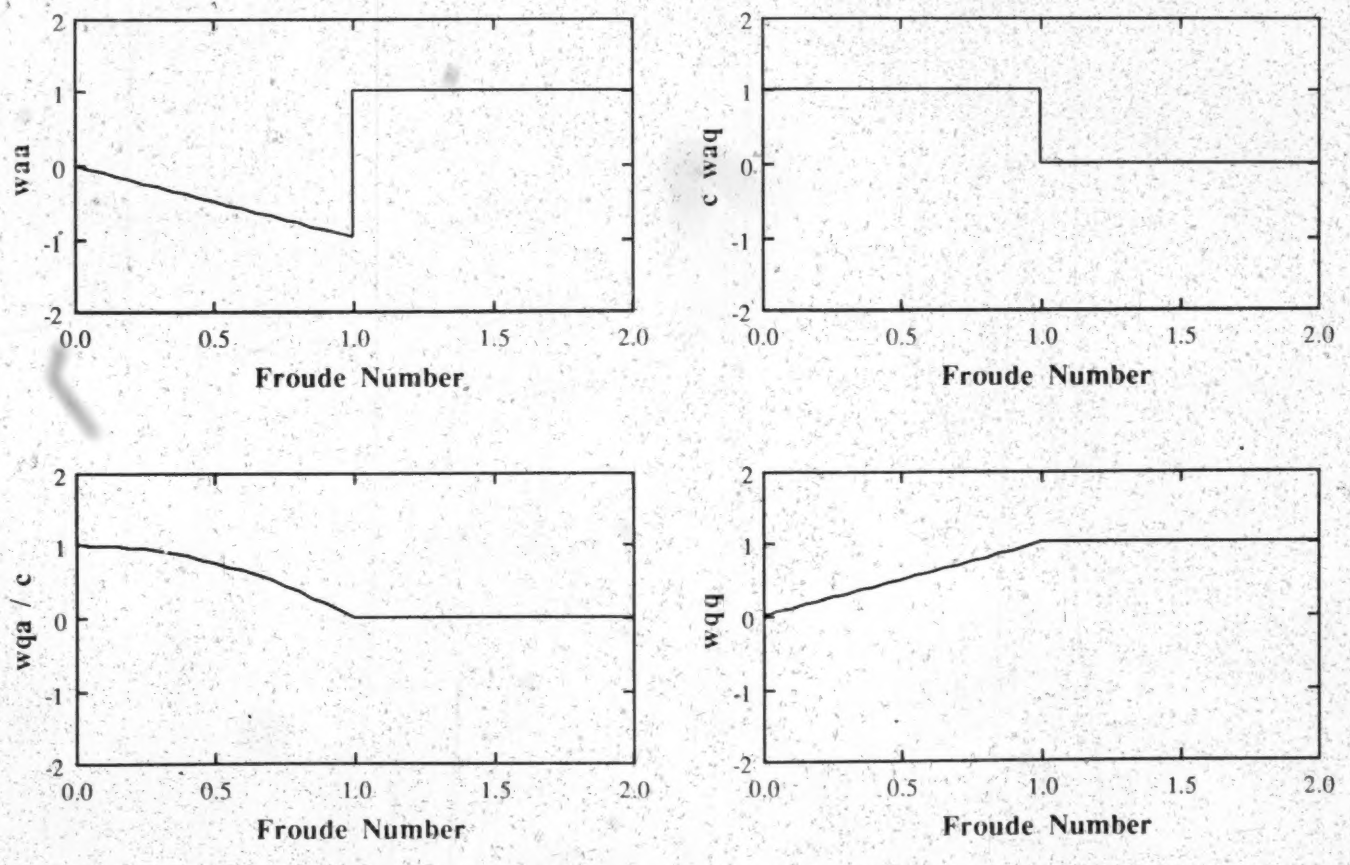


Figure 3.8 Variation of [W] with **F** for the CDG scheme.

4. LINEAR STABILITY ANALYSIS

4.1 INTRODUCTION

4.1.1 Background

The best criteria for selecting a suitable numerical method for solving the St. Venant equations, would be to assess the ability to accurately model the real life flow situations which are of concern. As man's involvement, exploitation and control of water resources increases, one can expect an ever increasing variety of flow situations to become of interest. For transient problems this might be the friction dominated wave typical of flood routing problems, or a very dynamic flow situation such as might be encountered when an ice jam releases. Unfortunately, comprehensive field data is not widely available, particularly for extreme events where costs and safety concerns are often prohibitive. Therefore, the engineer rarely has a means of achieving such a comparison. Furthermore, where data is available it must often be extrapolated for calibration purposes, an unreliable practice.

As introduced in Chapter 2, analytical solutions can provide a first level of comparison. Many potential numerical schemes may be excluded from further consideration if they are unable to provide stable and accurate solutions to the simplified equations. This introduces the most common indicators used in assessing computational methods: stability; convergence; and consistency. *Stability* is a measure of the reaction of a particular numerical scheme to a small perturbation introduced into the solution, for example due to truncation error. In the worst case these small errors accumulate exponentially, rapidly destroying the solution. (Liggett and Cunge, 1975) *Consistency* infers that the discrete solution converges to the solution of the continuous equations as Δx and Δt go to zero, that is as the discretization in space and time is refined. *Convergence* requires that the numerical calculations approach a solution of the discrete equations. According to Lax's Equivalence Theorem (Richtmyer and Morton, 1967) stability is the necessary and sufficient condition for convergence of a consistent method.

In this chapter the results of a linear stability analysis were used to examine the amplification and phase characteristics of the various numerical methods investigated in this study for the linearized St. Venant equations. The type of analysis used here may be described as a Fourier (or von Neumann) type (Roache, 1972). Because this is a linear analysis, it does not predict instabilities associated with nonlinearities in the problem. In addition, it does not consider the effects of gradient boundary conditions (Roache, 1972). Nevertheless, such an analysis is a valuable tool not only as a basis for the comparison of various numerical schemes but also to determine the appropriate discretization of a problem for a particular method.

Both extremes were considered in this analysis, that is: shallow water gravity waves in which friction and slope terms become negligible; and kinematic waves, where these terms dominate.

4.1.2 Implementation

The first step in the analysis was to determine an equivalent finite difference formulation for each of the numerical schemes to be investigated. For those numerical methods considered in this study, this finite difference representation of the non-dimensional, linear equations

$$\frac{\partial\{\phi\}}{\partial t} + \begin{bmatrix} 0 & 1 \\ 1 - \mathbf{F}^2 & 2\mathbf{F} \end{bmatrix} \frac{\partial\{\phi\}}{\partial x} + \begin{bmatrix} 0 & 0 \\ \dfrac{-3\beta}{H'\,\mathbf{F}^2} & \dfrac{2\beta}{\mathbf{F}^2} \end{bmatrix} \{\phi\} = \{0\} \qquad [2.122]$$

for

$$\{\phi\} \equiv \begin{Bmatrix} y \\ q \end{Bmatrix} \qquad [2.119]$$

may be generalized to the form:

$$a_{i,1}\, y_{j-1}^{n+1} + a_{i,2}\, y_{j}^{n+1} + a_{i,3}\, y_{j+1}^{n+1} + a_{i,4}\, q_{j-1}^{n+1} + a_{i,5}\, q_{j}^{n+1} + a_{i,6}\, q_{j+1}^{n+1} =$$

$$a_{i,7}\, y_{j-1}^{n} + a_{i,8}\, y_{j}^{n} + a_{i,9}\, y_{j+1}^{n} + a_{i,10}\, q_{j-1}^{n} + a_{i,11}\, q_{j}^{n} + a_{i,12}\, q_{j+1}^{n} \qquad [4.1]$$

in which,

$$y \equiv \Delta H \equiv \text{ depth perturbation} \qquad [4.2]$$

$$q \equiv \Delta(UH) \equiv \text{ discharge perturbation (per unit width of channel)} \qquad [4.3]$$

and i=1,2 (ie. one set of coefficients for each of the two equations).

Each Fourier component of the solution may assume the form (Roache, 1972):

$$\{\phi\}^{n+1}(x) = \{\Phi\}^{n+1} \exp(i\,\sigma\,x) \qquad [4.4]$$

where $\{\Phi\}^{n+1}$ represents the solution amplitude at time (n+1),

$$\{\Phi\}^{n+1} \equiv \begin{Bmatrix} Y^{n+1} \\ Q^{n+1} \end{Bmatrix} \qquad [4.5]$$

$$i \equiv \sqrt{-1} \qquad [4.6]$$

and σ represents the component wave number, such that,

$$\sigma \equiv \frac{2\pi}{L} \qquad [4.7]$$

where L is the wavelength:

$$L \equiv N\Delta x \qquad [4.8]$$

Therefore, N is the number of discretizations per wavelength.

The next step is to define the "amplification matrix", [G], such that:

$$\{\Phi\}^{n+1} = [G]\,\{\Phi\}^{n} \qquad [4.9]$$

where [G] describes both the amplification and phase characteristics of the numerical method. It may be obtained by substituting equation [4.4] into equation [4.1]:

$$\begin{aligned}
&a_{i,1}\,\mathbf{Y}^{n+1}\exp(-i\,\sigma\Delta x) + a_{i,2}\,\mathbf{Y}^{n+1} + a_{i,3}\,\mathbf{Y}^{n+1}\exp(i\,\sigma\Delta x) + \\
&\quad a_{i,4}\,\mathbf{Q}^{n+1}\exp(-i\,\sigma\Delta x) + a_{i,5}\,\mathbf{Q}^{n+1} + a_{i,6}\,\mathbf{Q}^{n+1}\exp(i\,\sigma\Delta x) = \\
&\quad a_{i,7}\,\mathbf{Y}^{n}\exp(-i\,\sigma\Delta x) + a_{i,8}\,\mathbf{Y}^{n} + a_{i,9}\,\mathbf{Y}^{n}\exp(i\,\sigma\Delta x) + \\
&\quad a_{i,10}\,\mathbf{Q}^{n}\exp(-i\,\sigma\Delta x) + a_{i,11}\,\mathbf{Q}^{n} + a_{i,12}\,\mathbf{Q}^{n}\exp(i\,\sigma\Delta x)
\end{aligned} \qquad [4.10]$$

and then collecting to obtain

$$b_{i,1}\,\{\mathbf{Y}\}^{n+1} + b_{i,2}\,\{\mathbf{Q}\}^{n+1} = b_{i,3}\,\{\mathbf{Y}\}^{n} + b_{i,4}\,\{\mathbf{Q}\}^{n} \qquad [4.11]$$

where:

$$b_{i,1} = a_{i,1}\exp(-i\,\sigma\Delta x) + a_{i,2} + a_{i,3}\exp(i\,\sigma\Delta x) \qquad [4.12]$$

$$b_{i,2} = a_{i,4}\exp(-i\,\sigma\Delta x) + a_{i,5} + a_{i,6}\exp(i\,\sigma\Delta x) \qquad [4.13]$$

$$b_{i,3} = a_{i,7}\exp(-i\,\sigma\Delta x) + a_{i,8} + a_{i,9}\exp(i\,\sigma\Delta x) \qquad [4.14]$$

$$b_{i,4} = a_{i,10}\exp(-i\,\sigma\Delta x) + a_{i,11} + a_{i,12}\exp(i\,\sigma\Delta x) \qquad [4.15]$$

This may be expressed in matrix notation as:

$$\begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} \begin{Bmatrix} \mathbf{Y}^{n+1} \\ \mathbf{Q}^{n+1} \end{Bmatrix} = \begin{bmatrix} b_{1,3} & b_{1,4} \\ b_{2,3} & b_{2,4} \end{bmatrix} \begin{Bmatrix} \mathbf{Y}^{n} \\ \mathbf{Q}^{n} \end{Bmatrix} \qquad [4.16]$$

so that,

$$[G] = \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix}^{-1} \begin{bmatrix} b_{1,3} & b_{1,4} \\ b_{2,3} & b_{2,4} \end{bmatrix} \qquad [4.17]$$

The two eigenvalues of the matrix [G] correspond to the two characteristic propagation velocities. These eigenvalues have the form:

$$r\, e^{i\Theta} \qquad [4.18]$$

where, r represents the magnification factor or the "algorithmic damping" coefficient and Θ indicates the total phase shift ,that is:

$$\Theta \equiv \sigma\, \Delta x \qquad [4.19]$$

where Δx is the distance moved by the wave. Determination of the phase shift error requires consideration of the expected (analytical) phase shift. That is,

$$\Theta \equiv \Theta_{anal.} + \Theta_{error} \qquad [4.20]$$

Considering a time period Δt, between "n" and "n+1", a disturbance would be expected to move a distance of "$c\,\Delta t$" where c represents the wave propagation velocity. This facilitates definition of the "relative celerity" as:

$$\text{relative celerity} \equiv \frac{\Theta}{\Theta_{anal.}} = \frac{\Theta}{-\sigma\,\Delta x} = \frac{\Theta}{-\sigma\, c\,\Delta t} = \frac{\Theta}{-\sigma\,\Delta x\, \dfrac{c\,\Delta t}{\Delta x}} = \frac{\Theta}{-\dfrac{2\pi}{N}\, C} \qquad [4.21]$$

in which C represents the Courant number, defined in equation [3.24]. Thus, definition of the relative celerity and algorithmic damping depends only on the coefficients in equation [4.1] which are particular to a given numerical scheme. In addition, an implicit method would introduce θ into equations [4.9] and [4.17].

4.2 SHALLOW WATER GRAVITY WAVES

4.2.1 Introduction

The shallow water gravity wave equations may be represented in non-dimensional linearized form as:

$$\frac{\partial\{\phi\}}{\partial t} + \begin{bmatrix} 0 & 1 \\ 1 - \mathbf{F}^2 & 2\mathbf{F} \end{bmatrix} \frac{\partial\{\phi\}}{\partial x} = \{0\} \qquad [4.22]$$

where,

$$\mathbf{F} \equiv \frac{U'}{\sqrt{gH'}} \qquad [2.116]$$

in which H' and U' represent the uniform depth and velocity, respectively.

For the fully dynamic case, the solution propagates at wave speeds of $\left| U' \pm \sqrt{gH'} \right|$. Thus, equation [4.21] may be expanded to:

$$\text{relative celerity} \equiv \frac{\Theta}{\pm \frac{2\pi}{N} C} \qquad [4.23]$$

where, the negative sign corresponds to the progressive $\left(U' + \sqrt{gH'}\right)$ wave and the plus sign is used for the regressive $\left(U' - \sqrt{gH'}\right)$ wave. It should be noted, however, that the Courant number is based upon the progressive wave velocity in all cases. Therefore, the results for the progressive waves were found to be independent of the Froude number for all of the methods.

4.2.2 The Box Finite Difference Scheme

The Box Finite Difference scheme was initially examined for a semi-implicit formulation ($\theta = 0.5$), to allow for direct comparison with the finite element methods. Details of the implementation of the linear stability analysis for this method are provided in

Appendix D. As seen in Figure 4.1 (progressive and regressive waves, **F** = 0.0) this scheme is entirely non-dissipative for all values of the Courant number, C. In addition, the relative celerity is essentially 1.0 (indicating a zero phase error) for C = 1. For lower values of C, this scheme would propagate the shorter wavelengths too quickly, while at larger values of C the shorter wavelengths would be propagated too slowly.

Figure 4.2 shows the results for the regressive waves at **F** = 0.5 and 2.0. Again the scheme was found to be completely non-dissipative. For Courant numbers of 0.5, 1, and 2 it was found that the shorter wavelengths would be propagated too quickly. A value of C = 5 lead to the opposite effect.

Figure 4.3 illustrates the results for the regressive waves at **F** = 0.99 (approximating critical flow). In this case, it was found that the shorter wavelengths would be propagated too quickly for all Courant numbers tested (up to 5) and, again, it was completely non-dissipative.

This difference in the phase errors between the progressive and regressive waves indicates that the selection of an appropriate Courant number would require a sacrifice for one of the two. In addition, the non-dissipative nature of the Box Finite Difference scheme has been found to be marginally stable at a θ of 0.5 for non-linear problems. Since a value of $\theta = 0.6$ has been suggested as an optimum for this scheme (Fread,1988) to assure stability, this was investigated next.

Figure 4.4 illustrates the results of the analysis for a value of $\theta = 0.6$ (progressive and regressive waves, **F** = 0.0). It is obvious from the figure that the increased implicitness has a significant dissipative effect which increases with the Courant number. However, the relative celerity is unaffected by the increased value of θ.

Figure 4.5 shows the results for the regressive waves at **F** = 0.5 and 2.0, while the results for **F** = 0.99 are shown in Figure 4.6. Both figures illustrate that the dissipative effect is reduced in comparison with the progressive wave, particularly near critical flow. Again, as seen in both figures, the relative celerity was unaffected.

The cost associated with the use of the Box Finite Difference method can be seen in a comparison of the amplitude and phase errors shown in Figures 4.4 and 4.5. At a value of θ = 0.6 the scheme might be run at a Courant number of 1 to minimize the phase error for the progressive wave. However, since U+c varies throughout the domain, this would be extremely difficult to achieve. In addition, more than 10 nodes per wavelength would be required to avoid both excessive damping of the progressive wave and regressive wave phase error.

4.2.3 The Bubnov-Galerkin Method

The Bubnov-Galerkin method was examined for both a semi-implicit (θ = 0.5) and a fully implicit (θ = 1.0) formulation. Details of the implementation of the linear stability analysis for this method are provided in Appendix D. As seen in Figure 4.7 (progressive and regressive waves, **F** = 0.0) this scheme, like the Box Finite Difference method, is entirely non-dissipative for all values of the Courant number, C. In addition, this scheme tends to propagate the shorter wavelengths too slowly, with this effect increasing significantly for the larger values of C.

Figure 4.8 shows the results for the regressive waves at **F** = 0.5 and 2.0. As in the case of the progressive waves, the scheme was found to be completely non-dissipative. In addition, this method would tend to propagate the shorter wavelengths too slowly, though the regressive waves would have less phase error than the progressive ones.

The results for the regressive waves at **F** = 0.99 can be seen in Figure 4.9. In this case, it was found that both the algorithmic damping and the relative celerity were independent of the Courant number, at least for those values tested (up to 5). Very short wavelengths would be propagated too quickly, and the scheme was again entirely non-dissipative.

This non-dissipative nature of the Bubnov-Galerkin method causes this method to be vulnerable to spurious oscillations. This can lead to the complete deterioration of the solution in the non-linear case where numerous investigators have observed the accumulation of 2Δx waves. As in the Box Finite Difference scheme, there is a temptation to increase the level of implicitness to introduce some algorithmic damping. To illustrate the effect of such an approach, a value of $\theta = 1.0$ was investigated next.

Figure 4.10 illustrates the effect of increasing the value of θ to 1.0 for both the progressive and regressive waves at a Froude number of 0. It is obvious from the figure that the increased implicitness has a significant dissipative effect which increases with the Courant number. However, the 2Δx wave remain undamped indicating that this method would not benefit from such an approach. Unlike the Box Finite Difference scheme, the relative celerity is significantly reduced by the increased value of θ.

As seen in Figures 4.11, regressive waves at **F** = 0.5 and 2.0 would experience somewhat less algorithmic damping that progressive waves, though again the 2Δx waves would be undamped. In addition, the relative celerity would be reduced slightly by the increased implicitness. On the other hand, the results for **F** = 0.99, shown in Figure 4.12 are identical to that observed in Figure 4.9, for the semi-implicit formulation.

As discussed in Chapter 3, the poor performance of the Bubnov-Galerkin method has lead many investigators to the conclusion that the finite element method is unsuitable for open channel flow problems. The non-dissipative nature of the scheme explains the tendency for this scheme to deteriorate into nonsense as 2Δx waves accumulate as a result of aliasing in non-linear problems. However, this method, like the Box Finite Difference method, serves as a benchmark for comparison in considering the various finite element schemes discussed next.

4.2.4 The Taylor-Galerkin Method

Details of the implementation of the linear stability analysis for the Taylor-Galerkin scheme are provided in Appendix D. As this is an explicit scheme, only a value of $\theta = 0.0$ was considered. In addition, due to the reduced stability range, a lower Courant number of 0.25 was used in addition to the values of 0.5, 1, 2 and 5 run for all other (implicit) methods.

Figure 4.13 illustrates the results of the linear stability analysis for both progressive and regressive waves at $\mathbf{F} = 0$. For $C = 0.25$ and 0.5 this scheme displays the highly desirable selective damping of the very short wavelengths. In addition, the method displays remarkably good phase accuracy at $C = 0.25$. However, a significant overestimation of the phase speed would result with the selection of a Courant number of 0.5. As expected, this scheme displays the reduced stability range discussed in Chapter 3. Results for $C > 0.577$ are unstable and display poor phase accuracy as well.

As seen in Figure 4.14 both $C = 0.25$ an 0.5 perform well for the regressive wave at $\mathbf{F} = 0.5$ and 2.0, particularly in terms of the relative celerity, though the results are somewhat less dissipative. It is interesting to note that this figure shows the regressive wave to be stable and more accurately propagated that the progressive wave at $C = 1.0$.

The results for the regressive waves at $\mathbf{F} = 0.99$ can be seen in Figure 4.15. In this case, it was found that both the algorithmic damping and the relative celerity were independent of the Courant number, except at the very small wavelengths. Both the algorithmic damping and relative celerity are, in fact, essentially equal to 1.0 for the regressive wave for critical flow.

4.2.5 The Petrov-Galerkin Method

Details of the implementation of the linear stability analysis for the Petrov-Galerkin schemes are provided in Appendix D. As in the case of the Bubnov-Galerkin method, these implicit schemes were examined for both a semi-implicit ($\theta = 0.5$) and a fully implicit ($\theta = 1.0$) formulation.

4.2.5.1 The Dissipative-Galerkin Method

This analysis was run using a value of ω of $2/\sqrt{15}$ ($\omega \approx 0.5$) as given by equation [3.83]. As reported by Katopodes (1984), this method also displays the highly desirable selective damping of the high frequency wavelengths, as observed for the Taylor-Galerkin method. However, unlike the Taylor-Galerkin scheme, the Dissipative-Galerkin method is stable for all values of the Courant number tested, as seen in Figure 4.16 for the progressive and regressive waves at **F** = 0.0 (Katopodes, 1984). Examination of the phase characteristics, however, indicates that a Courant number of 0.5 is clearly appropriate for this scheme in order to assure good phase accuracy, as this scheme tends to underestimate the phase speed at larger values of C.

Figure 4.17 shows the results for the regressive waves at **F** = 0.5 and 2.0. As in the case of the progressive waves, the scheme was found to be selectively dissipative for the shorter wavelengths, though the regressive wave would be less damped than the progressive waves. For example, at $2\Delta x$ the algorithmic damping is 0.84 for the regressive wave compared to 0.13 for the progressive wave, when C = 0.5.

The results for the regressive waves at **F** = 0.99 can be seen in Figure 4.18. In this case, it appeared that the Dissipative-Galerkin method had no effect at critical flow conditions as the results are the same as those for the Bubnov-Galerkin scheme for both the algorithmic damping (non-dissipative) and the relative celerity.

Figure 4.19 illustrates the significant effect of increasing the value of θ to 1.0 for both the progressive and regressive waves at a Froude number of 0. It is obvious from the figure that the increased implicitness has a significant dissipative effect which increases with the Courant number. Similar results were observed for **F** = 0.5 and 2.0 as shown in Figure 4.20. Again, results for **F** = 0.99 show no effect due to increasing θ or due to the Dissipative-Galerkin method.

4.2.5.2 The Characteristic-Dissipative-Galerkin Method

To examine the sensitivity of the CDG scheme to the upwinding coefficient, it was first run for a Froude number of 0.5 and three values of ω, 0.25, 0.5 and 1.0. The results for a Courant number of 0.5 are shown for the progressive and regressive waves in Figures 4.22 and 4.23, respectively. Corresponding results for C = 1.0 are shown in Figures 4.24 and 4.25, respectively. For the progressive waves at ω = 0.25, the damping is most selective. Results for ω = 0.5 and 1.0 are not that different from each other. On the other hand, the phase accuracy of the scheme is better at ω = 0.5 than 0.25 and the results for the regressive wave show similar tendencies. For this study it was decided to study ω = 0.25 as a recommended value for unsteady flow analysis. Therefore, the remainder of the CDG linear stability analysis was conducted using this value.

As Figure 4.26 illustrates, this method is particularly successful at damping the high frequency wavelengths for the progressive and regressive waves at **F** = 0.0 using a value of ω = 0.25. And, like the Dissipative-Galerkin scheme, a Courant number of 0.5 is appropriate for this scheme in order to assure good phase accuracy.

Figure 4.27 shows the results for the regressive waves at **F** = 0.5 and 2.0. As in the case of the progressive waves, the scheme was found to be selectively dissipative for the shorter wavelengths.

The results for the regressive waves at **F** = 0.99 can be seen in Figure 4.28. In this case, the scheme exhibits particularly selective damping near critical flow conditions while, unlike the Bubnov-Galerkin and Dissipative-Galerkin schemes, 2Δx wavelengths are propagated.

Figure 4.29 illustrates the effect of increasing the value of θ to 1.0 for both the progressive and regressive waves at a Froude number of 0. The increased implicitness has a significant dissipative effect which increases with the Courant number. Similar results were observed for **F** = 0.5 and 2.0 and **F** = 0.99, as shown in Figures 4.30 and 4.31, respectively.

Because the CDG method was formulated for symmetric systems of hyperbolic equations, in consideration of orthogonal upwinding, this method was tested for the symmetric form of the equations [2.60] to see if there would be a significant cross-upwinding effect expected. These results are shown in Figures 4.32 (progressive and regressive waves, **F** = 0) and 4.33 for **F** = 0.5 and 2.0. As the results were not significantly different from the non-symmetric form of the equations, it was expected that this method could be applied to the non-symmetric system without expecting significant adverse cross-upwinding effects.

4.2.6 The Least-Squares Method

As this scheme was implemented as an equivalent Petrov-Galerkin method, details of the implementation of the linear stability analysis are included with the Petrov-Galerkin description in Appendix D. Again, this scheme was examined for both a semi-implicit ($\theta = 0.5$) and a fully implicit ($\theta = 1.0$) formulation.

Figure 4.34 illustrates the results of the linear stability analysis for both progressive and regressive waves at **F** = 0. This scheme displays selective damping of the very short wavelengths For C = 0.5 as well as good phase accuracy. However, results are not as good as the Courant number increases. A significant underestimation of the phase speed would result with the selection of a Courant number greater than 1.

As seen in Figure 4.35 both C = 0.25 an 0.5 exhibit good phase accuracy for the regressive wave at **F** = 0.5 and 2.0. However, there is significantly less damping of the short wavelengths at C = 0.5. It is interesting to note that this figure shows increased damping of the regressive wave at higher Courant numbers as well as improved phase accuracy.

The results for the regressive waves at **F** = 0.99 can be seen in Figure 4.36. In this case, it was found that both the algorithmic damping and the relative celerity were almost independent of the Courant number. The scheme was essentially non-dissipative for the regressive waves, though the phase accuracy was good.

Figures 4.37 to 4.39 illustrate the results for a $\theta = 1.0$. Results were typical of other methods in that a significant increase in dissipation was associated with the increase in implicitness.

4.2.7 Comparison of Methods

Figure 4.40 illustrates a comparison of the various methods investigated at a Froude number of zero for both the progressive and regressive wave (representative of the progressive waves at all other Froude numbers, as well). In this case, a value of 10 nodes per wavelength was selected so as to compare the algorithmic damping and relative celerity as functions of the Courant number. For the purpose of comparison, each method was run at its recommended level of implicitness. This was $\theta = 0.5$ for all implicit finite element methods, $\theta = 0$ for the Taylor-Galerkin scheme and $\theta = 0.6$ for the Box Finite Difference scheme. Ideally, the objective is to damp out only the shorter wavelengths while retaining good phase and amplitude accuracy on the longer ones. Therefore, the 'best' methods would be those that exhibited no algorithmic damping or phase error of these $10\Delta x$ waves.

As the figure shows, all of the implicit finite element methods display almost no dissipation of the $10\Delta x$ waves for Courant numbers up to 100. The Taylor-Galerkin scheme also displays this tendency within its stable range, although it is interesting to note the significant increase in amplification which occurs beyond $C = 1.0$. The Box Finite Difference scheme is also non-dissipative for a $10\Delta x$ wave up to a Courant number of about 0.5. However, it is undesirably overdissipative at $C = 1.0$.

The implicit finite element methods display good phase accuracy up to a Courant number of about 0.5, indicating this as an appropriate value for transient problems. The Taylor-Galerkin scheme, however, would be better run at a value of $C \leq 0.3$ to avoid propagating these wavelengths too quickly. The phase accuracy of the Box Finite Difference scheme indicates an interesting point about this method in that $10\Delta x$ waves would be propagated too quickly for $C \leq 1$ and too slowly for $C \geq 1$. This means that both good amplitude and phase accuracy could not be achieved with this method at this resolution. Therefore, it appears that the BFD scheme requires a finer mesh to achieve the same accuracy as the implicit finite element methods.

Figure 4.41 illustrates a similar comparison, this time for the regressive wave at a Froude number of 0.5. The tendencies are the same, although it is interesting to note the expanded stability range of the TG scheme as well as the improvement in performance of the BFD scheme compared to the progressive wave at **F** = 0.5 and the regressive wave at

$F = 0$ (both represented by Figure 4.40). Also important is the deteriorated phase accuracy performance of this last method. Again, both good amplitude and phase accuracy could not be achieved with this method at this resolution.

It was also desirable to compare the two most promising finite element schemes (DG and CDG) at a shorter wavelength. Figure 4.42 shows the amplitude and phase accuracy of these two methods for the regressive waves at a Froude number of 0.5, as a function of the Courant number for $4\Delta x$ waves. Results of the BFD method are also included for reference purposes. As the figure shows, the CDG schemed displays slightly more damping and marginally improved phase accuracy in comparison with the DG method.

As discussed in Chapter 3, these two methods are quite similar in their consideration of the progressive wave. The difference between the two lies in the handling of the regressive wave. This might explain the necessity for increasing the value of ω beyond that indicated by equation [3.85] when using the DG scheme in the vicinity of shocks (Katopodes, 1984). To examine this possibility, Figures 4.43 and 4.44 compare the amplitude and phase accuracy of the two methods at short wavelengths, for $C = 0.5$, $F = 0.5$ and $\theta = 0.5$. Also included in the plots are results for the DG scheme for increased values of the upwinding coefficient (ie. 1.5x indicates a value of ω 1.5 times that given by equation [3.85]). As seen in Figure 4.43, the DG scheme is more dissipative than the CDG for progressive waves (as expected given the value of ω was 0.25 for the CDG method). It also displays slightly better phase accuracy. Increasing the value of ω does not increase the damping on the very short wavelengths (less than 4 or 5) but does significantly affect the phase accuracy of the DG scheme.

Based on these results, it would not appear desirable, or for that matter even necessary, to increase the value of ω when using the DG method. This only becomes apparent when the regressive waves are considered. As Figure 4.44 shows, the CDG

($\omega = 0.25$) is more dissipative than the DG ($\omega \approx 0.5$) for short wavelengths. It also displays slightly better phase accuracy (as noted earlier). When the value of ω is increased by a factor of 1.5 in the DG method it performs nearly identically to the CDG scheme for the regressive wave. When ω is increased by a factor of 7.5, the short wavelength are further damped for the regressive wave while the phase accuracy is substantially reduced for both progressive and regressive waves for a wide range of wavelengths. This factor of 7.5 corresponds to that used by Katopodes (1984) in modeling an hydraulic jump.

4.3 KINEMATIC WAVES

4.3.1 Introduction

In most practical situations, even the most dynamic flows eventually progress towards a diffusive type of wave as an event propagates over a long reach. Therefore, consideration of only the shallow water gravity wave problem would examine only part of the problem in terms of this study's objective. This is of particular interest to methods like the DG and CDG which employ upwinding in an amount determined based on the dynamic terms in the equations. Therefore, it important to determine the consequences of employing these methods on waves that are not fully dynamic.

The non-dimensional equations may be represented in linearized form as:

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \frac{\partial\{\phi\}}{\partial t} + \begin{bmatrix} 0 & 1 \\ \left(\frac{1}{F^2} - 1\right) & 2 \end{bmatrix} \frac{\partial\{\phi\}}{\partial x} + \begin{bmatrix} 0 & 0 \\ -3\beta & 2\beta \end{bmatrix} \{\phi\} = \{0\} \qquad [2.118]$$

where,

$$\beta = \frac{L}{H' C_*^2} \qquad [2.117]$$

For the kinematic wave ($\beta = \infty$) this reduces to:

$$\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \frac{\partial\{\phi\}}{\partial t} + \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \frac{\partial\{\phi\}}{\partial x} + \begin{bmatrix} 0 & 0 \\ -3 & 2 \end{bmatrix} \{\phi\} = \{0\} \qquad [4.24]$$

which is independent of the Froude number. Consideration of this situation has the advantage of including only those terms neglected in the shallow water gravity wave case. For the purpose of this study, the kinematic wave problem was represented using the system of equations([4.24]) to gain insight into the expected behavior of a fully dynamic model on problems for which these terms become dominant.

4.3.2 Kinematic Wave Analysis

Figure 4.45 illustrates the amplitude and phase accuracy of the BFD scheme on equation [4.24] for $\theta = 0.6$. The results are identical to those obtained for the shallow water gravity wave problem (noting that the Courant number has been redefined, based upon the kinematic wave speed). Result for the BG method were also found to be the same as for the shallow water gravity wave case.

For the DG and CDG method, the stability analysis could not be run on equation [4.24] due to the dependence of the upwinding on the dynamic terms. Therefore, these methods were run on the complete dynamic equation [2.122] using a value of $\beta = 1000$ to approach the kinematic wave case. Figures 4.46 and 4.47 show the results for the DG and CDG schemes, respectively, at **F** = 0.5 and $\theta = 0.5$. Both methods display less damping of wavelengths greater than $5\Delta x$ than the BFD at $\theta = 0.6$ (Figure 4.4) though their phase accuracy is comparable. Both methods also exhibit selective damping of most short wavelengths. Of particular importance is the fact that both the DG and the CDG schemes are completely non-dissipative for the $2\Delta x$ waves. This means that an attempt to model a kinematic wave with these schemes could lead to an unstable solution. Fortunately, most practical situations are more diffusive than kinematic and, in addition, these waves tend to be quite long and therefore beyond the $2\Delta x$ size.

For comparative purposes, the results at $\beta = 1000$ are provided for the Taylor-Galerkin and the Least-Squares methods as well. As Figure 4.48 shows, at $C = 0.25$ the TG scheme is slightly more dissipative than it is for the fully dynamic case. In addition, these shorter wavelengths would be propagated too quickly. Of particular interest is the fact that the stability range of this scheme is reduced from the dynamic value of 0.577. It was determined to be 0.353 for values of β in excess of 100. For the Least-Squares method, Figure 4.49 shows that this method would be substantially less dissipative of $2\Delta x$ wave at $C = 0.5$ and 1.0 than in the fully dynamic case. In addition, it appears much less selective. Poorer phase accuracy is displayed, as well.

To examine the behavior of the CDG scheme further, the algorithmic damping for $2\Delta x$ waves was examined as a function of β. The results, shown in Figure 4.50, indicate that dissipation of these wavelengths would occur for $\beta < 1000$, particularly at $C = 0.5$ and 1.0. To illustrate this, the analysis for the CDG was run using a value of $\beta = 5$ (ie. a diffusive wave). The results, shown in Figure 4.51, verify that damping of $2\Delta x$ waves would occur. However, because these waves would be diffusive by nature, such an analysis gives no information on the accuracy of such a solution. Future research might investigate the analytical amplitude and phase characteristics of diffusive waves to facilitate such an evaluation.

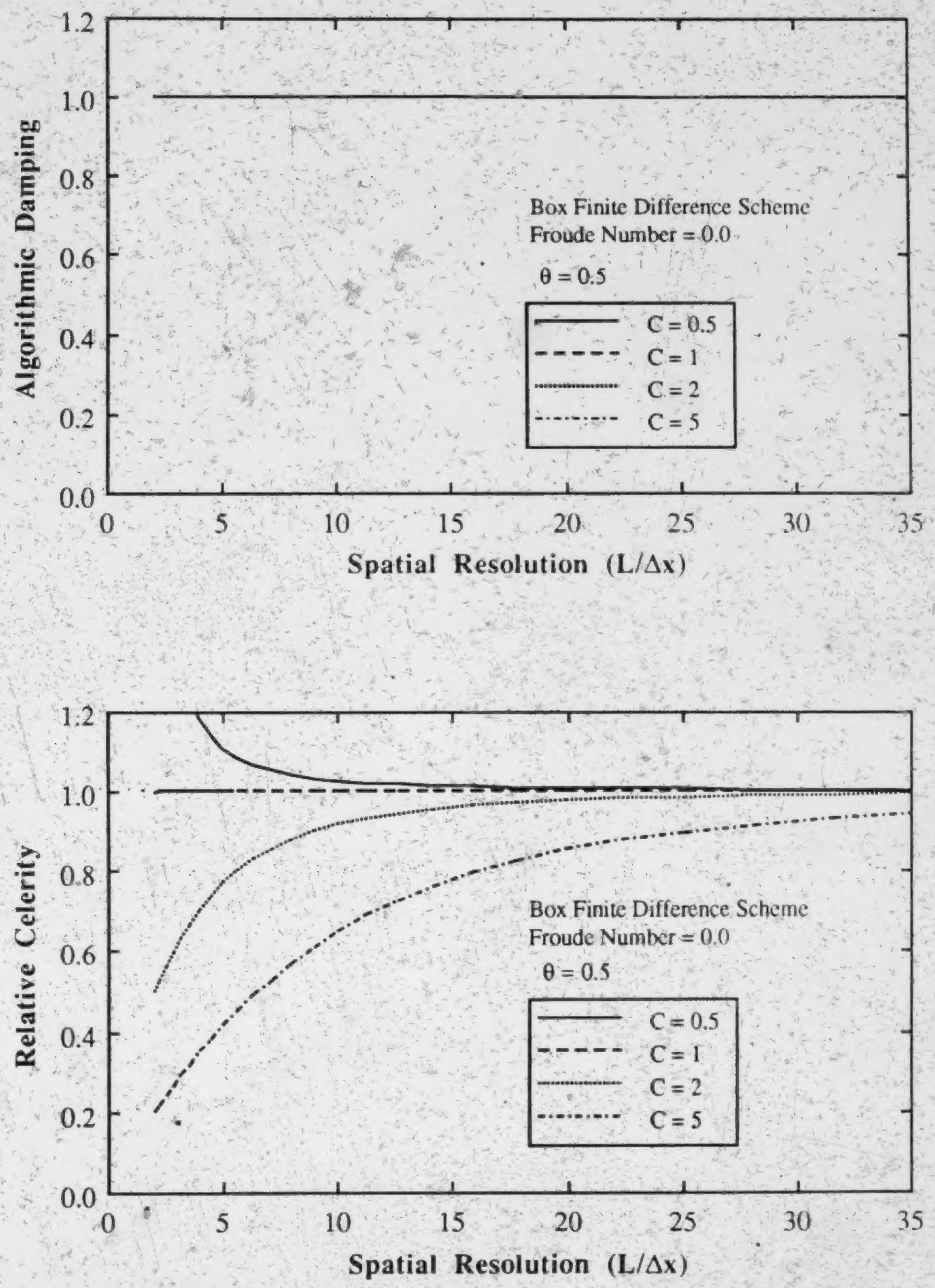


Figure 4.1 Results of stability analysis, Box Finite Difference Scheme, **F** = 0.0, θ = 0.5 (progressive and regressive waves).

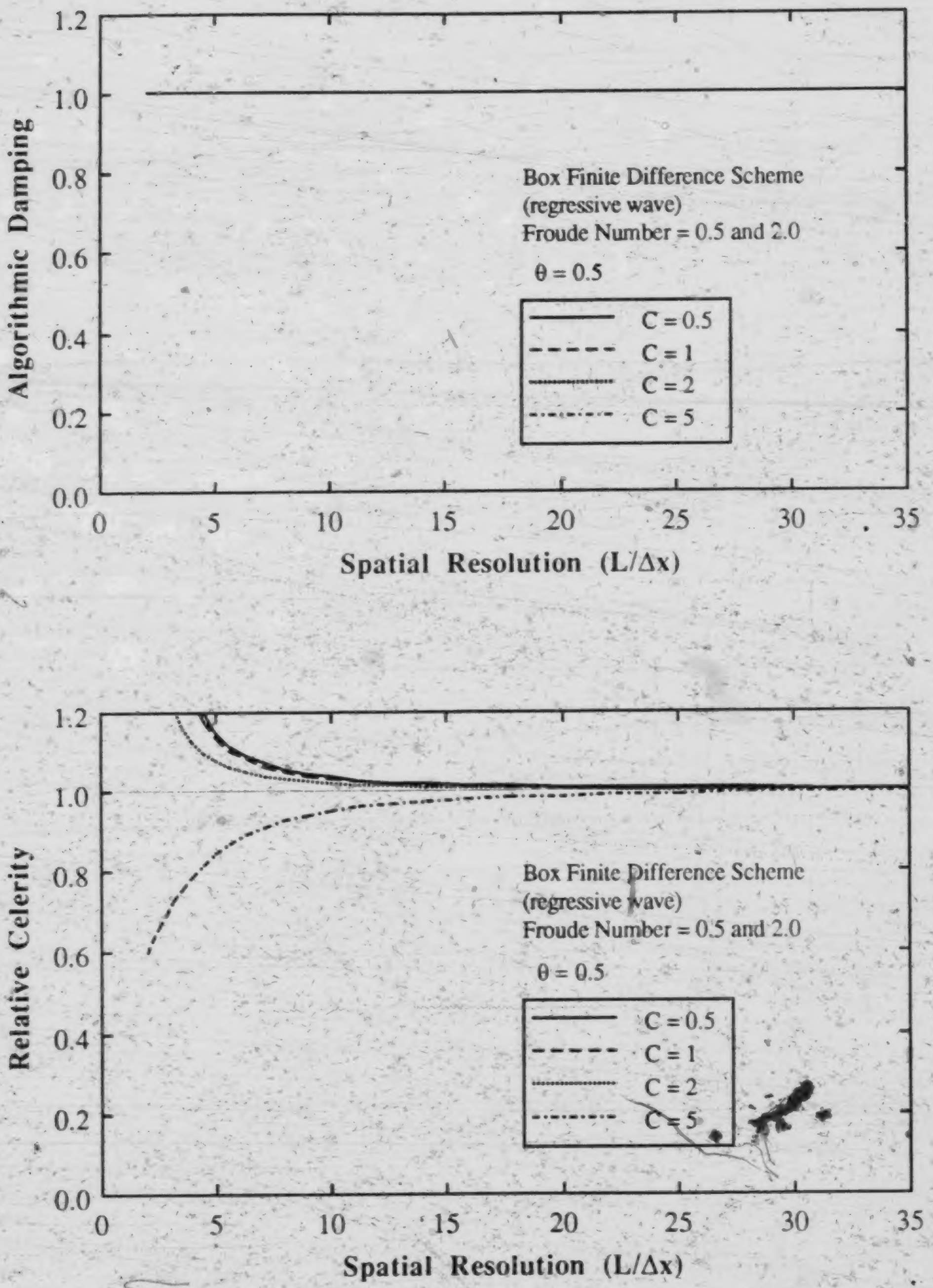


Figure 4.2 Results of stability analysis, Box Finite Difference Scheme, **F** = 0.5 and 2.0, θ = 0.5 (regressive wave).

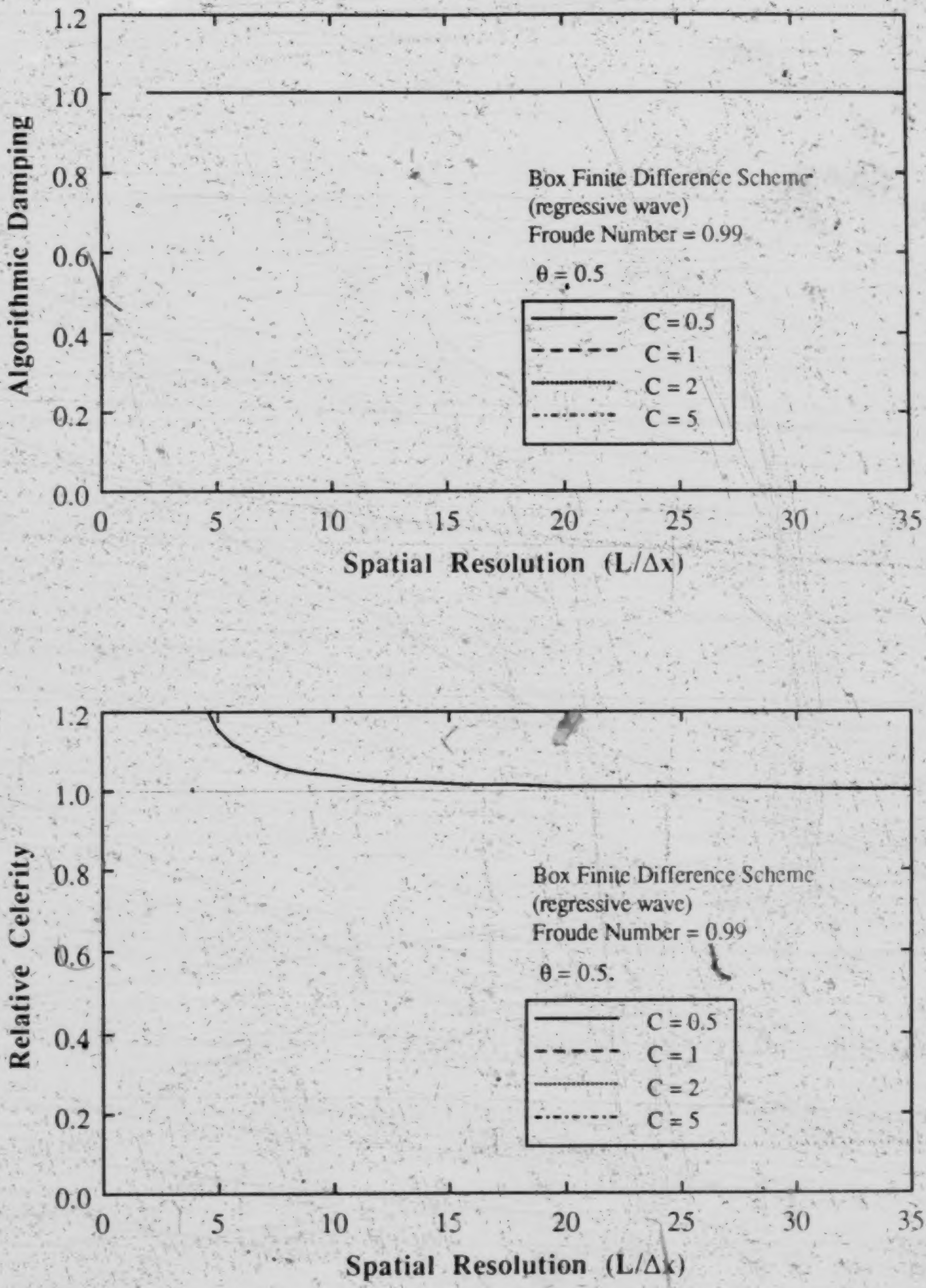


Figure 4.3 Results of stability analysis, Box Finite Difference Scheme, F = 0.99, θ = 0.5 (regressive wave).

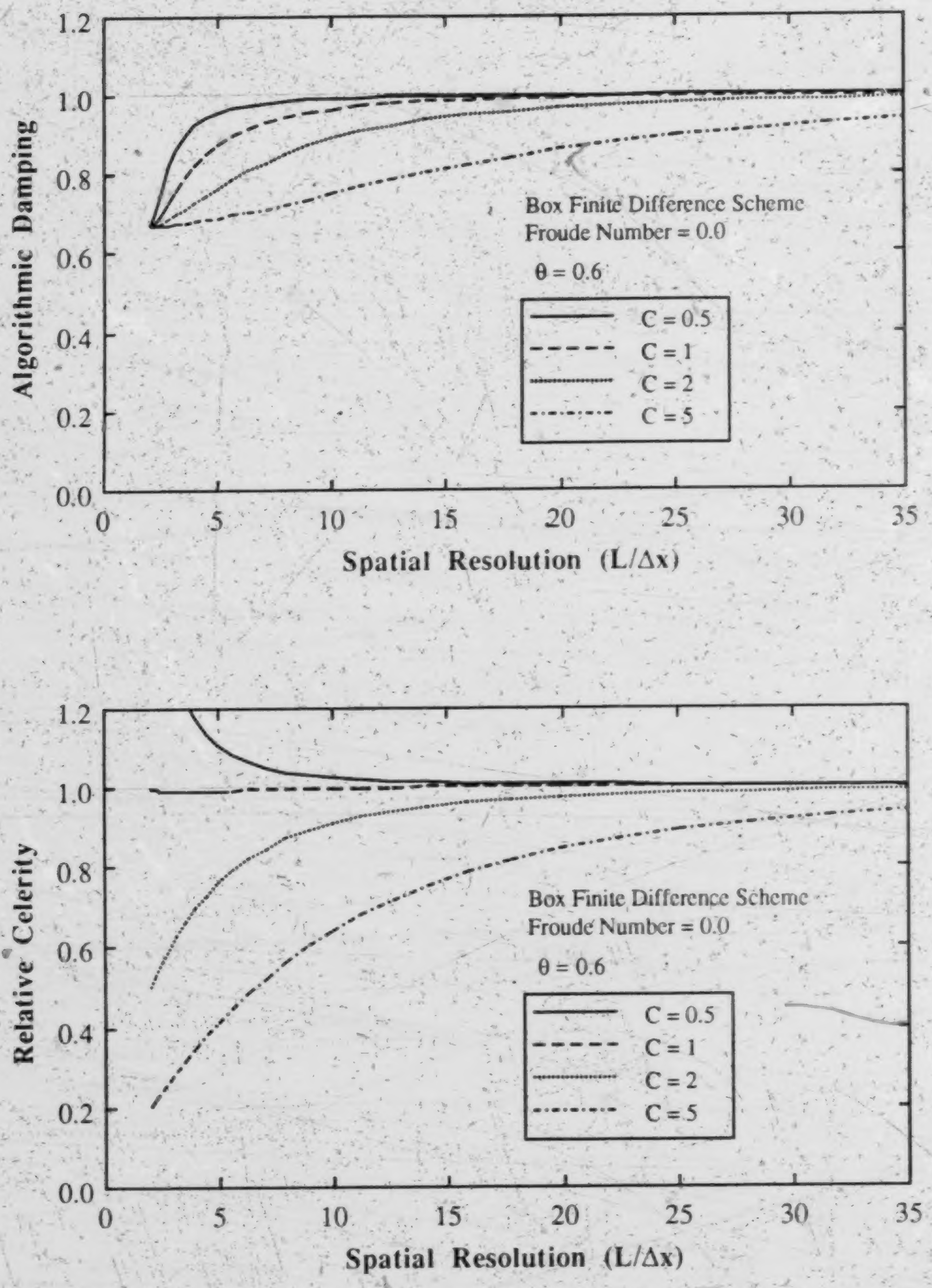


Figure 4.4 Results of stability analysis, Box Finite Difference Scheme, F = 0.0, θ = 0.6 (progressive and regressive waves).

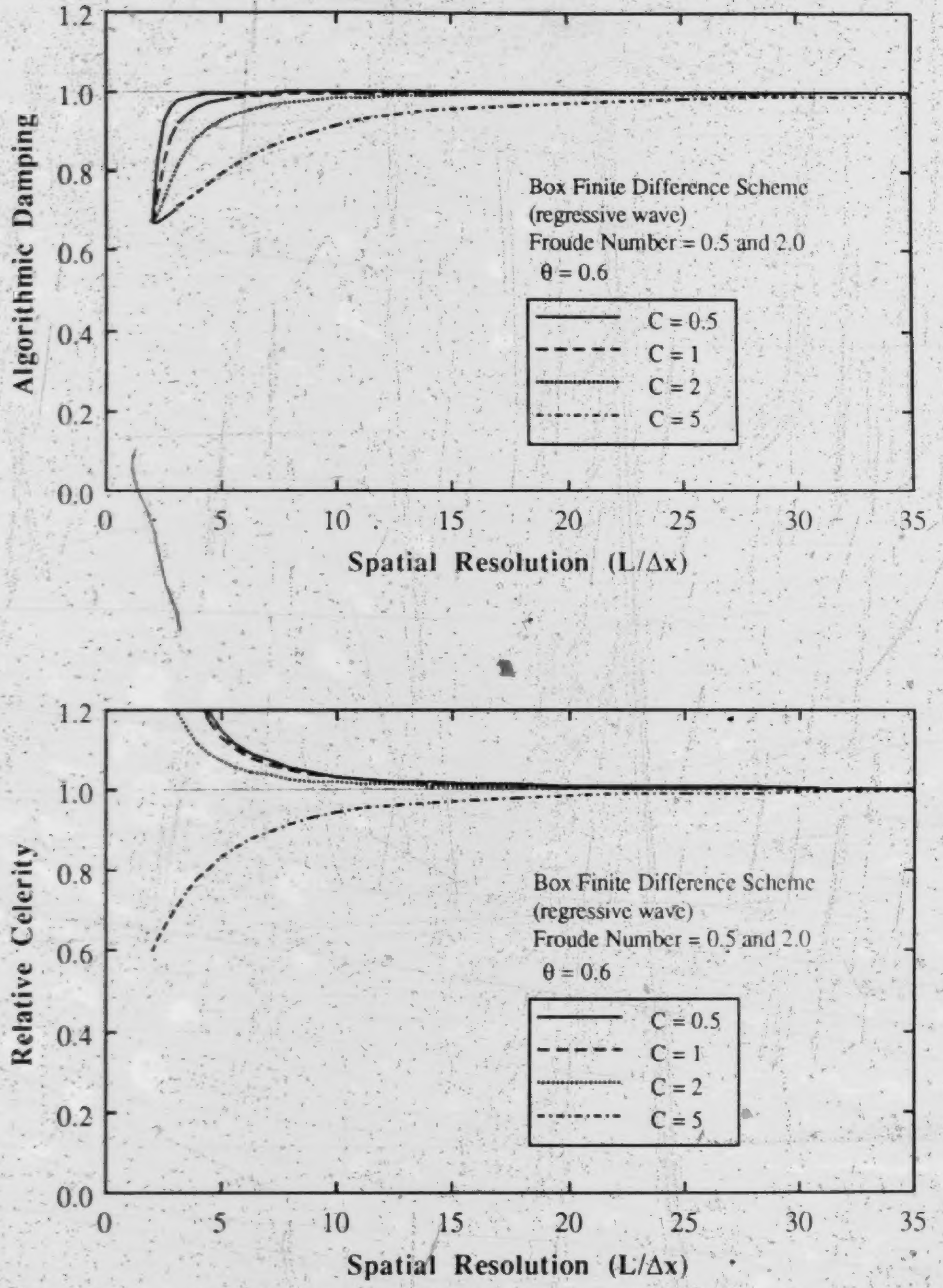
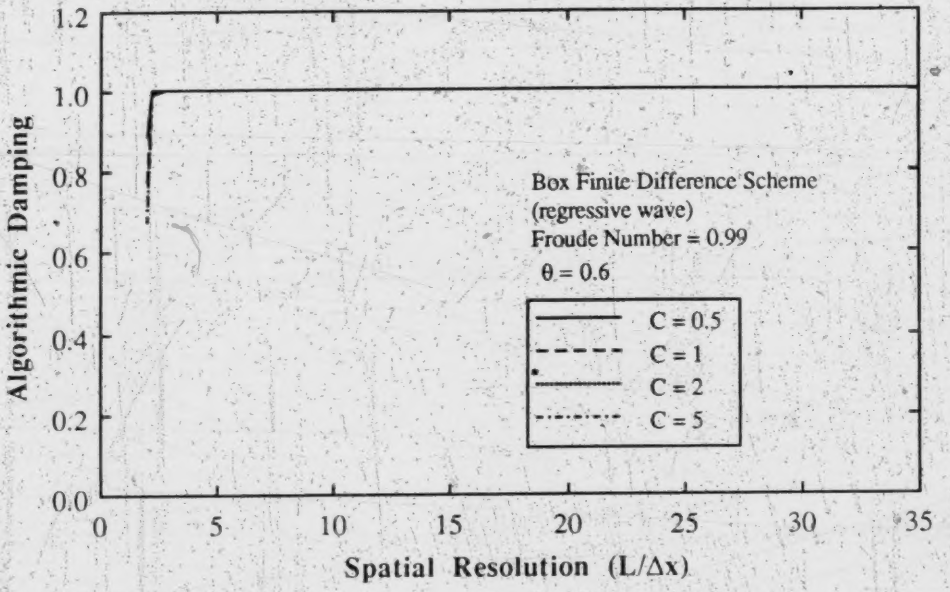


Figure 4.5 Results of stability analysis, Box Finite Difference Scheme, F = 0.5 and 2.0, θ = 0.6 (regressive wave).



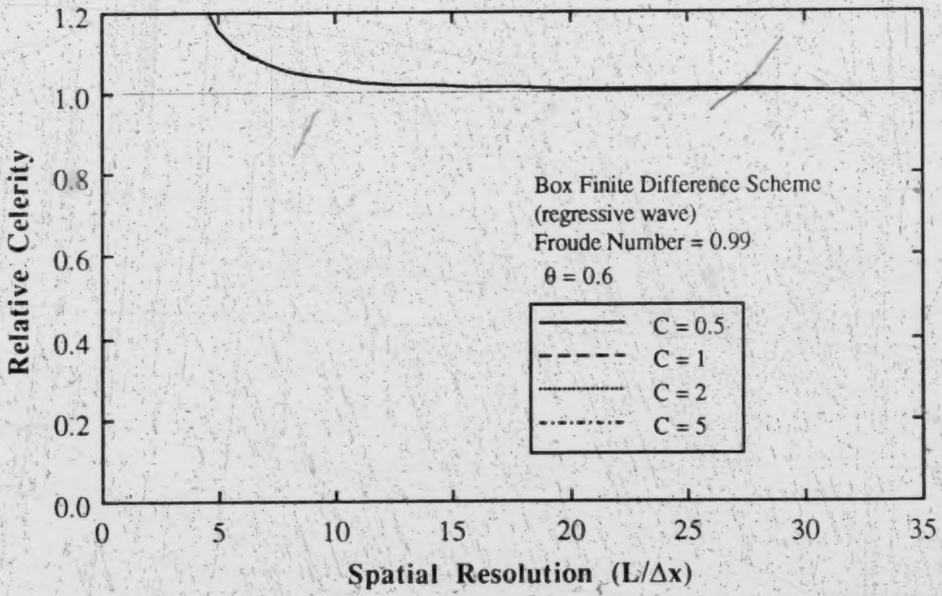


Figure 4.6 Results of stability analysis, Box Finite Difference Scheme, **F** = 0.99, θ = 0.6 (regressive wave).

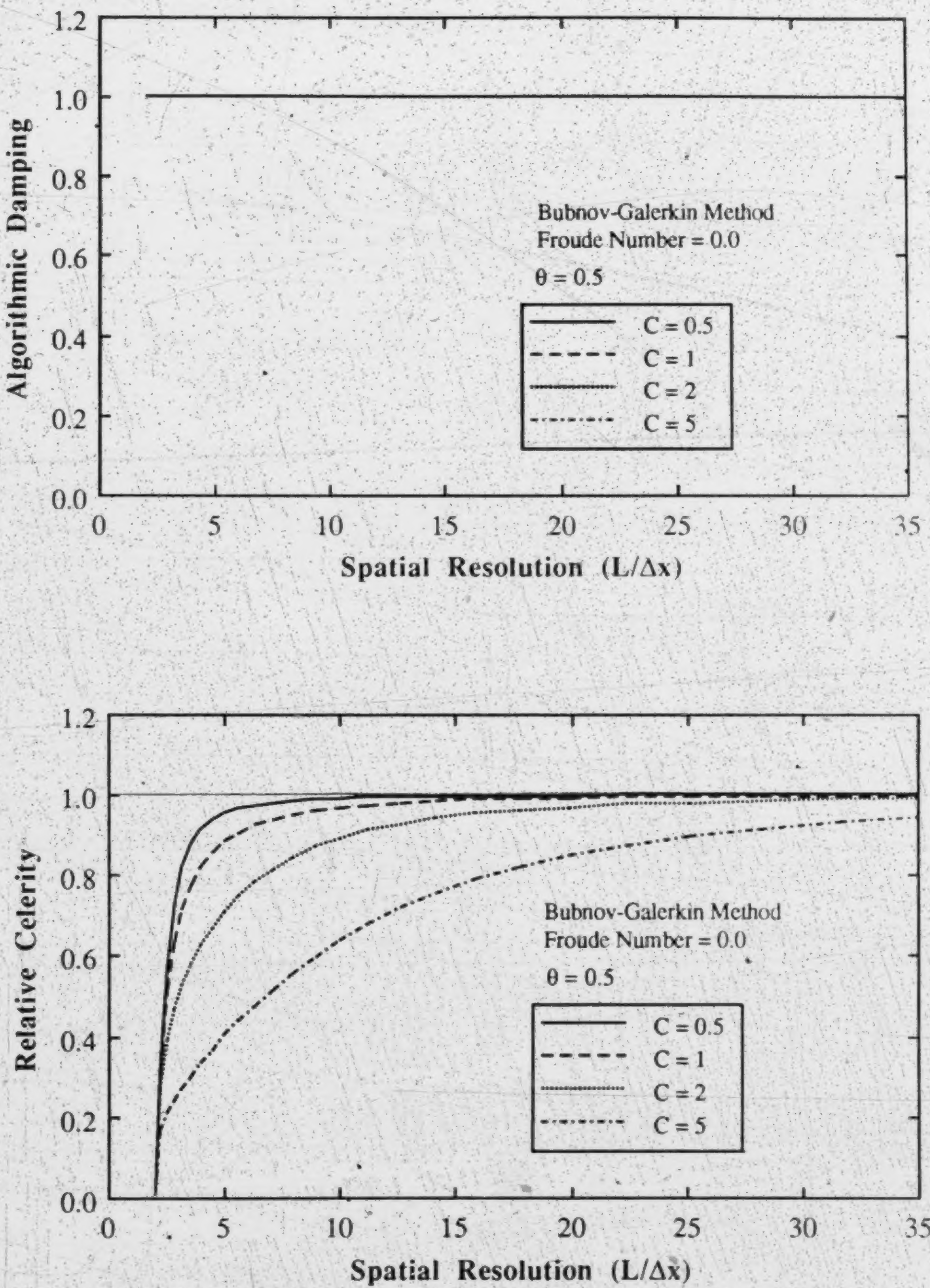
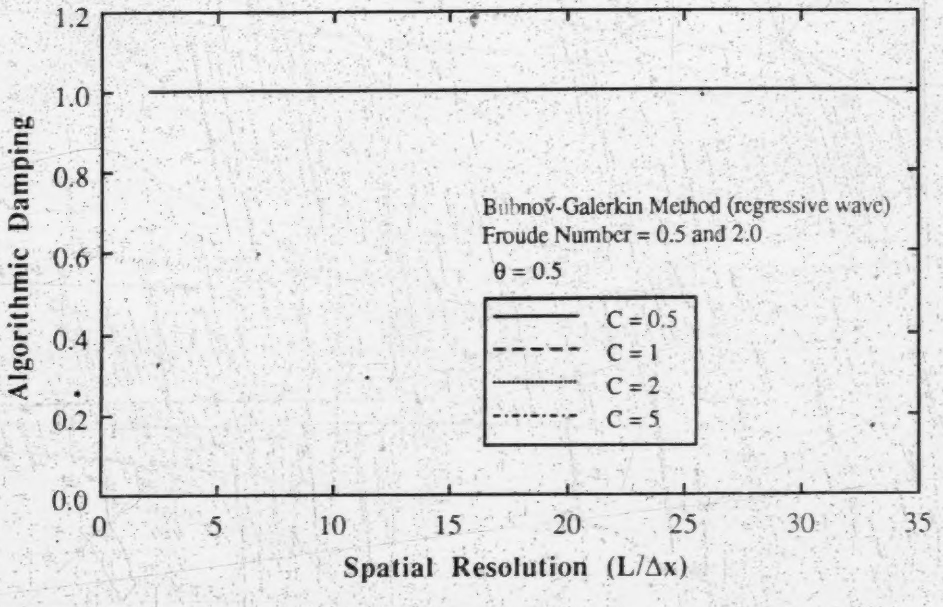


Figure 4.7 Results of stability analysis, Bubnov-Galerkin Method, F = 0.0, θ = 0.5 (progressive and regressive waves).



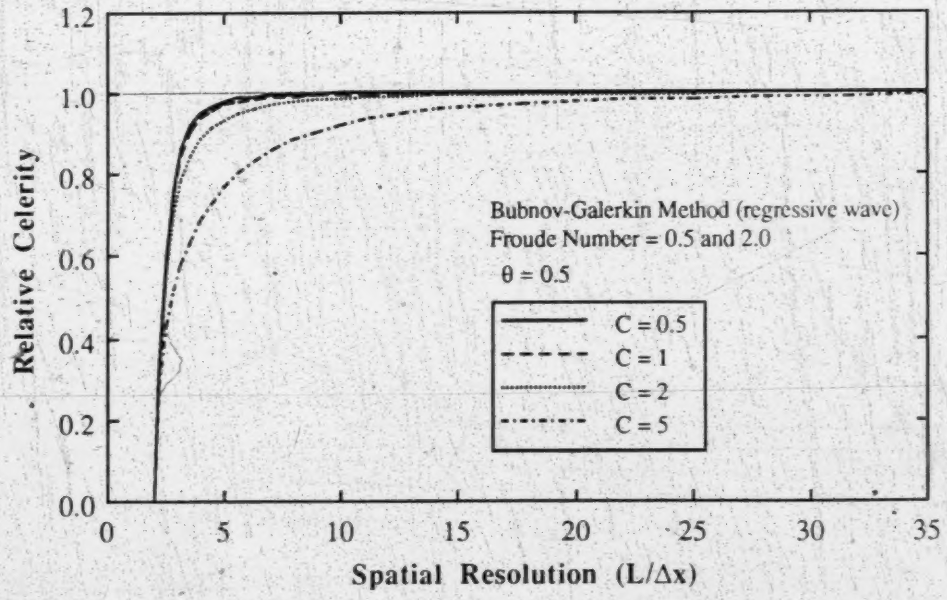
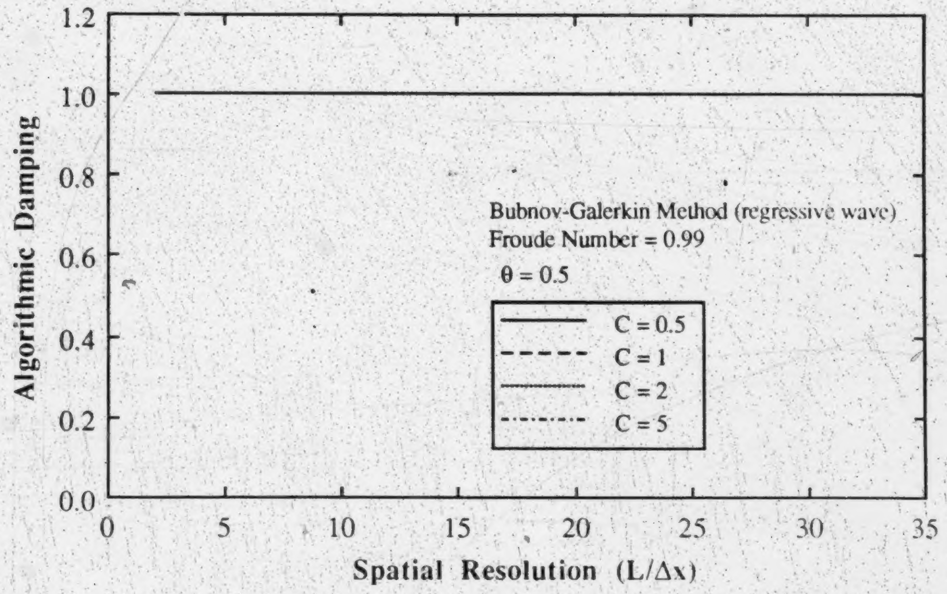


Figure 4.8 Results of stability analysis, Bubnov-Galerkin Method, **F** = 0.5 and 2.0, θ = 0.5 (regressive wave).



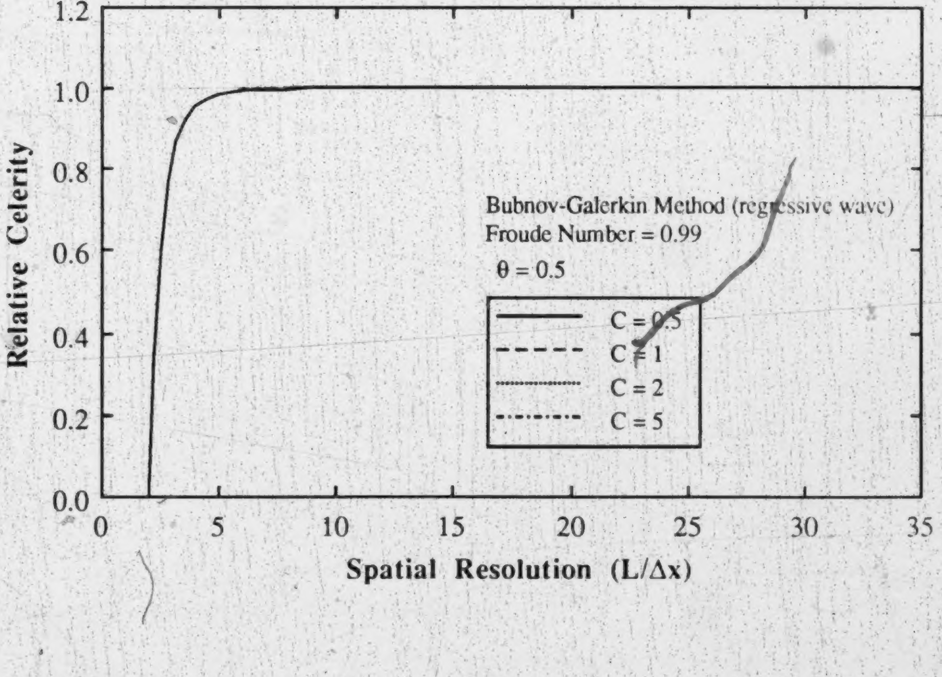


Figure 4.9 Results of stability analysis, Bubnov-Galerkin Method, **F** = 0.99, θ = 0.5 (regressive wave).

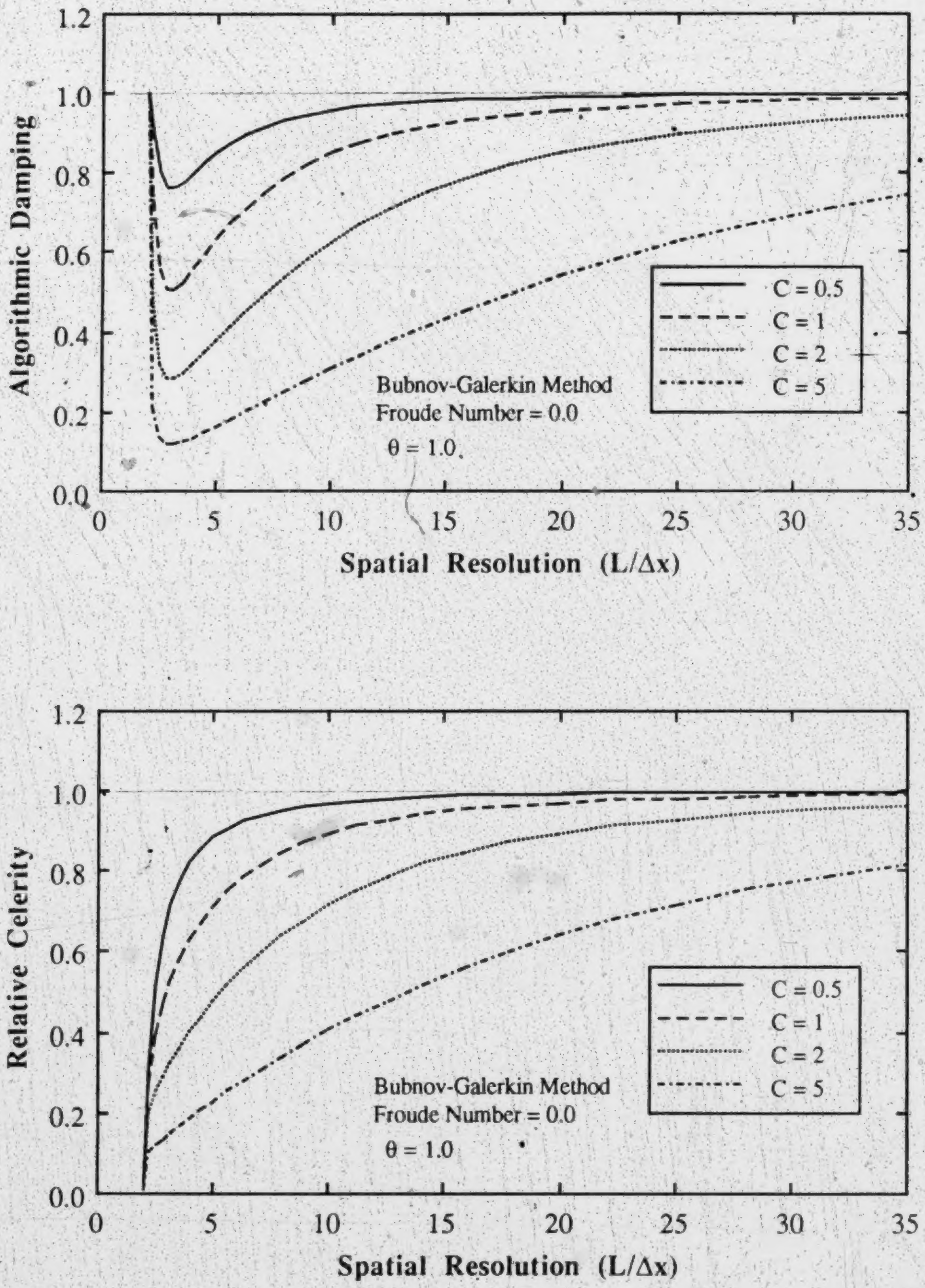
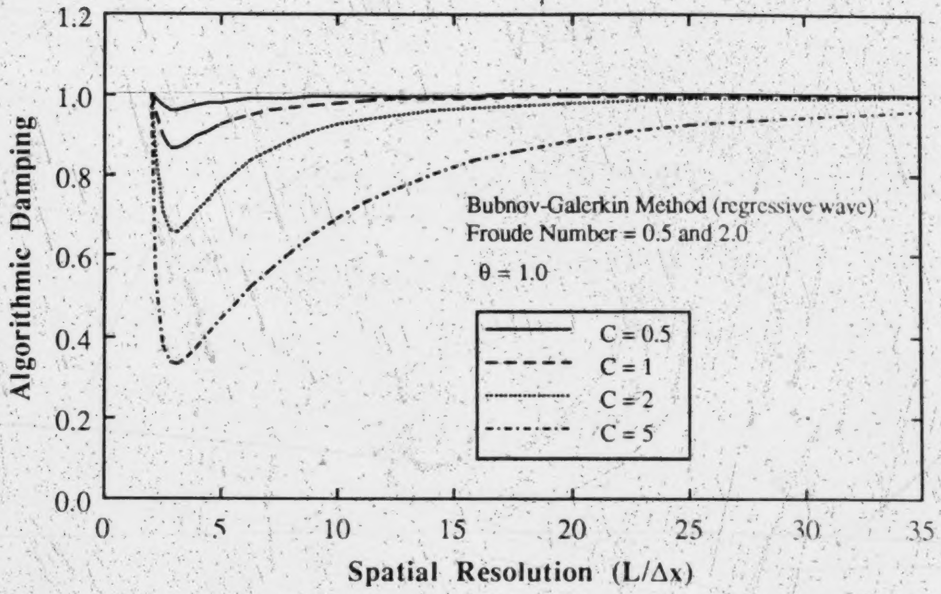


Figure 4.10 Results of stability analysis, Bubnov-Galerkin Method, **F** = 0.0, θ = 1.0 (progressive and regressive waves).



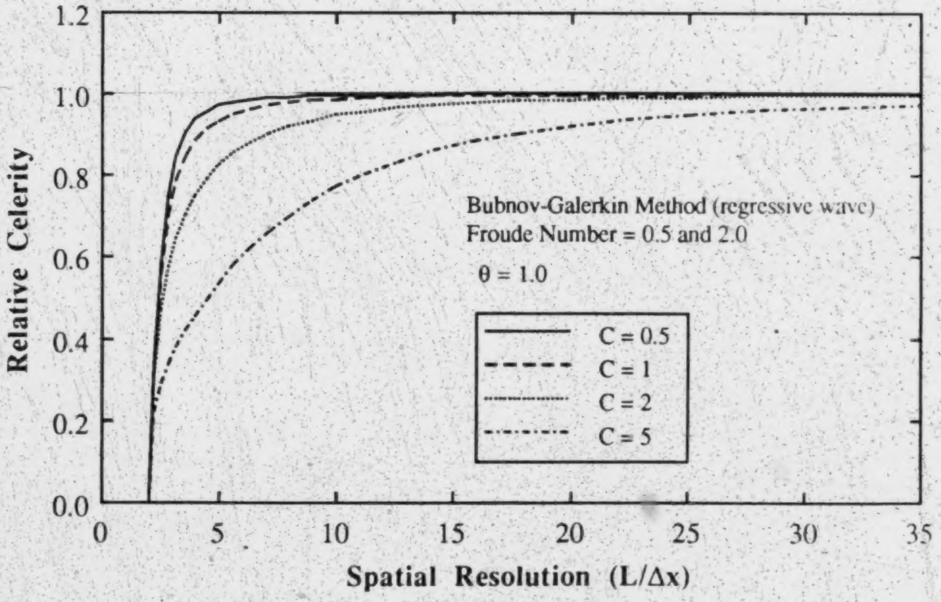
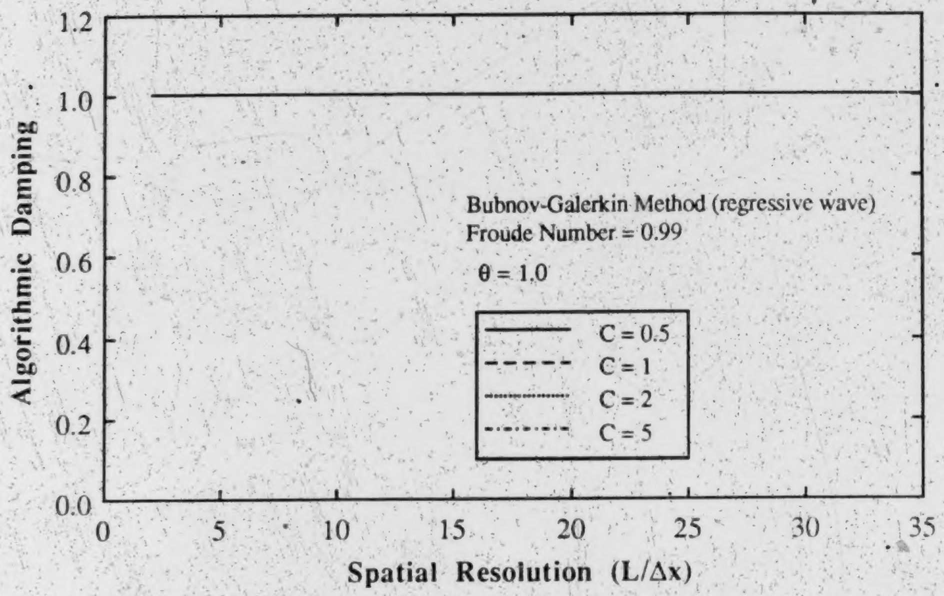


Figure 4.11 Results of stability analysis, Bubnov-Galerkin Method, F = 0.5 and 2.0, θ = 1.0 (regressive wave).



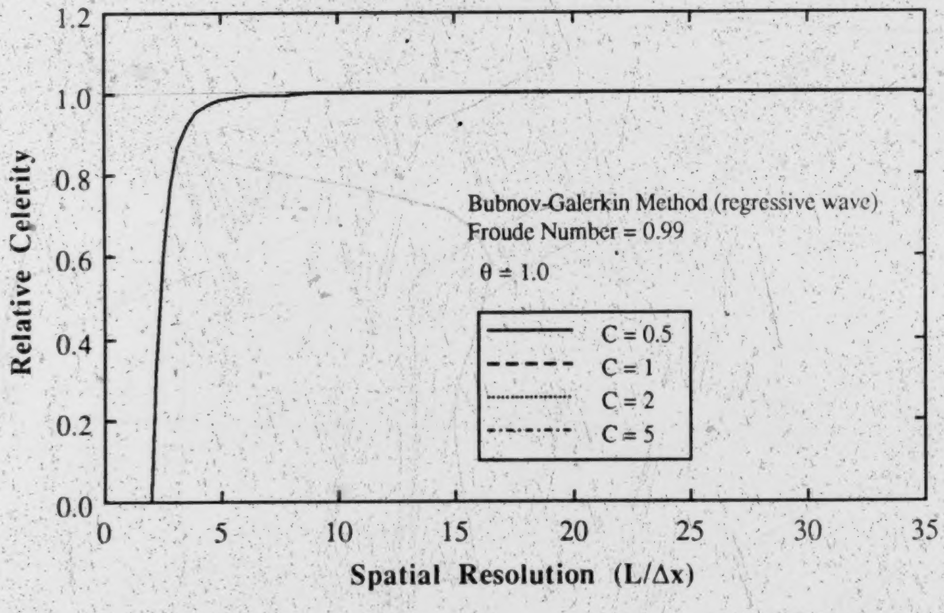
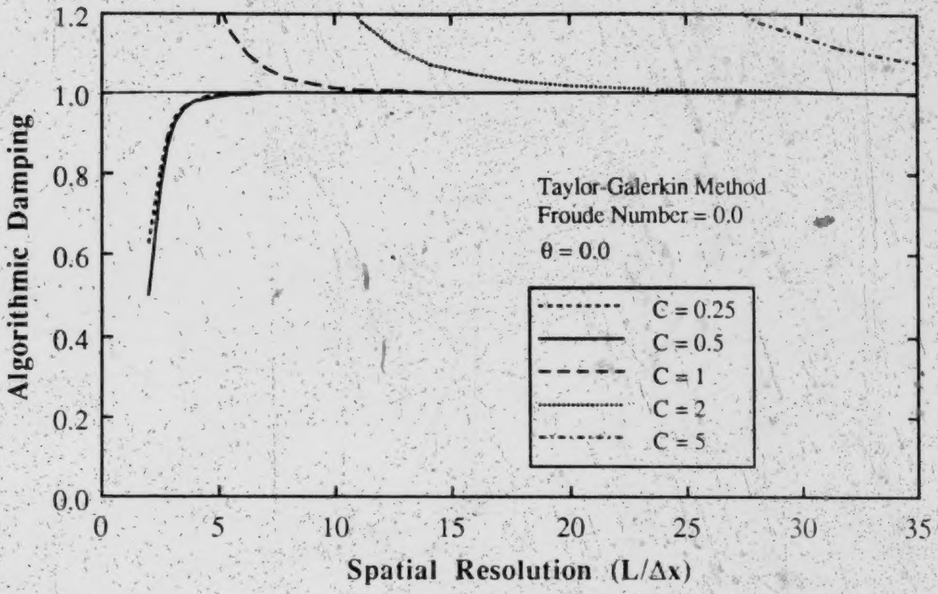


Figure 4.12 Results of stability analysis, Bubnov-Galerkin Method, **F** = 0.99, θ = 1.0 (regressive wave).



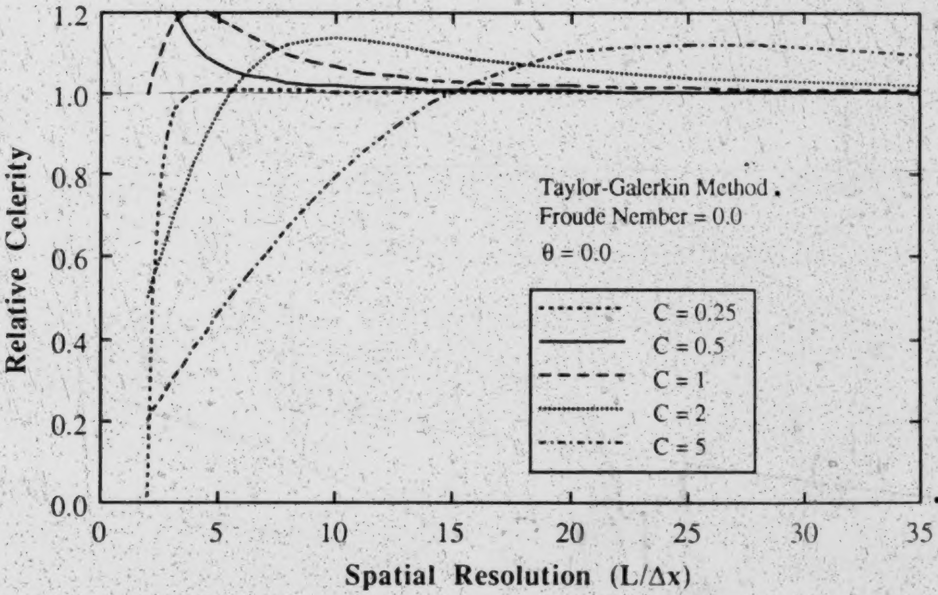
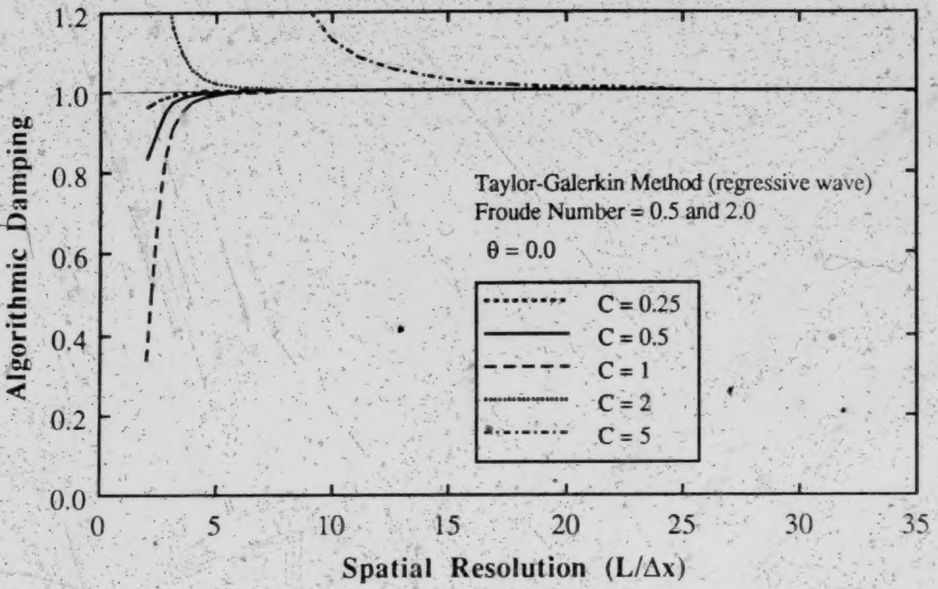


Figure 4.13 Results of stability analysis, Taylor-Galerkin Method, F = 0.0, θ = 0.0 (progressive and regressive waves).



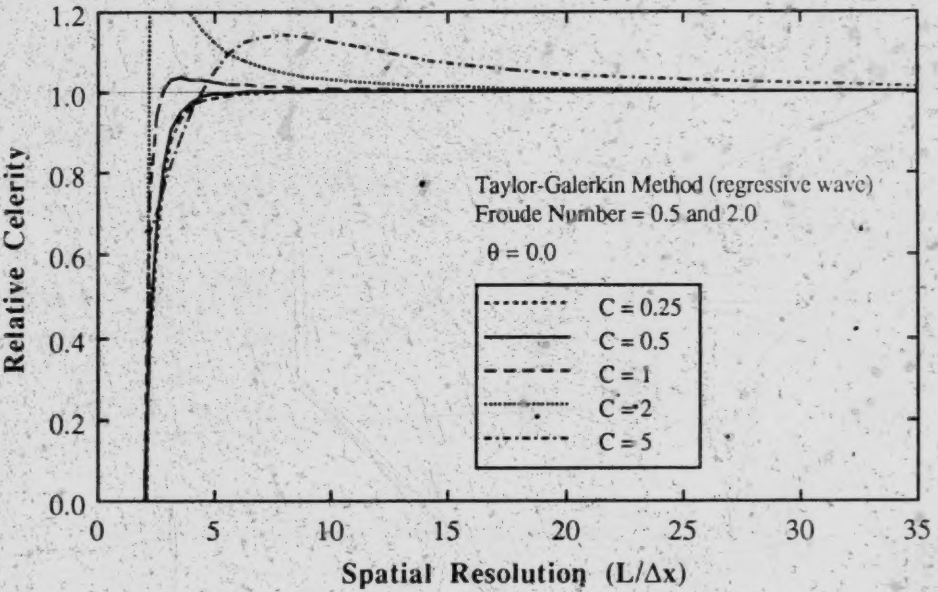
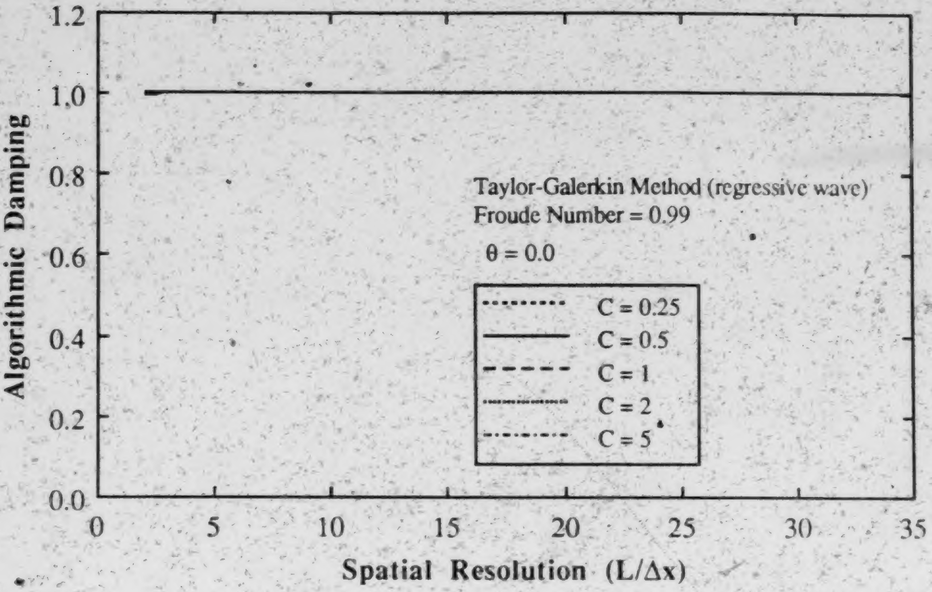


Figure 4.14 Results of stability analysis, Taylor-Galerkin Method, **F** = 0.5 and 2.0, θ = 0.0 (regressive wave).



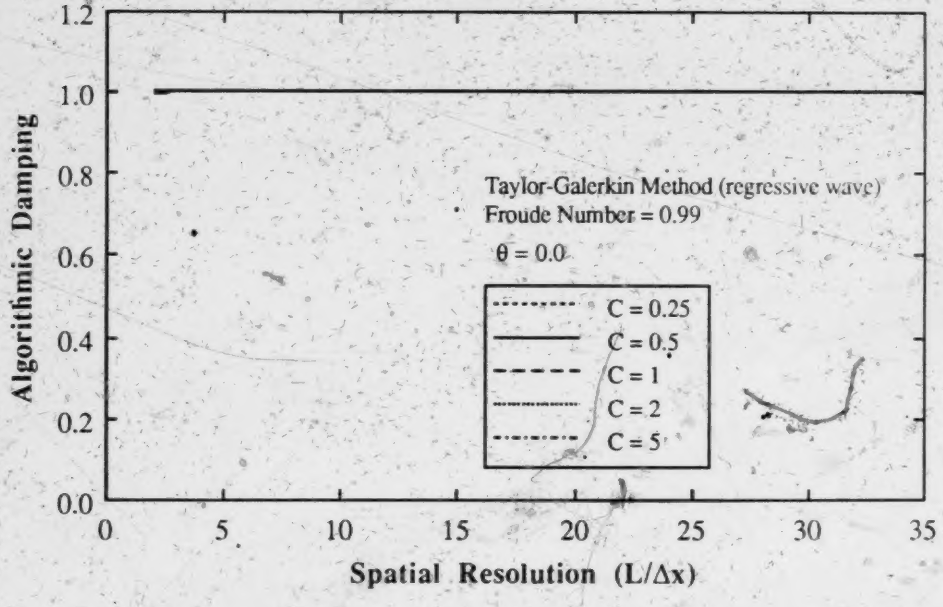
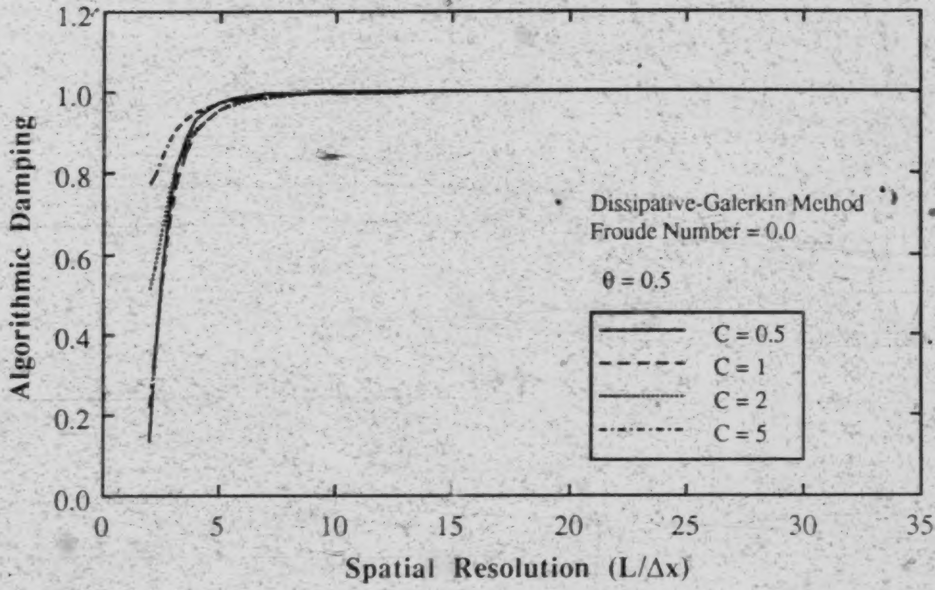


Figure 4.15 Results of stability analysis, Taylor-Galerkin Method, **F** = 0.99, θ = 0.0 (regressive wave).



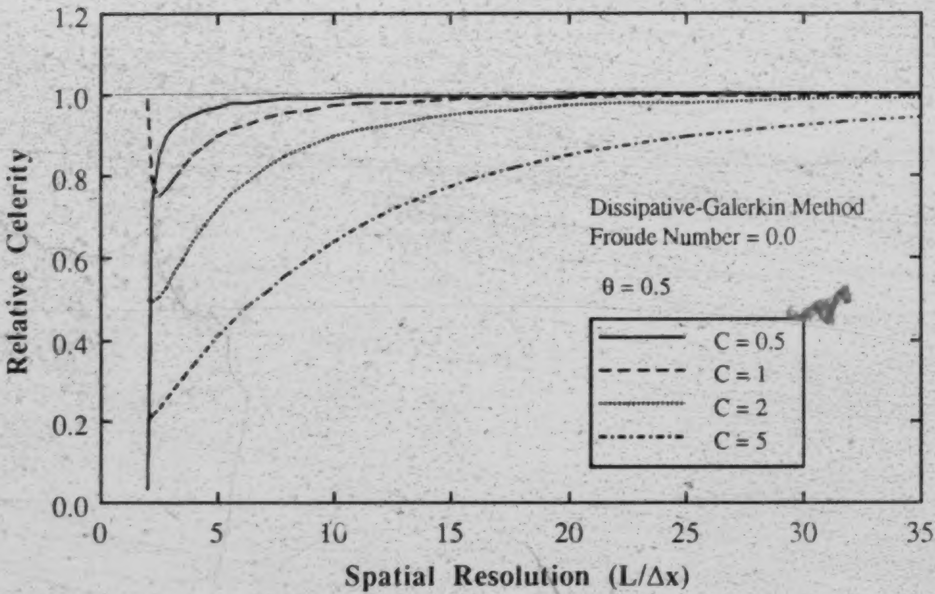


Figure 4.16 Results of stability analysis, Dissipative-Galerkin Method, **F** = 0.0, θ = 0.5 (progressive and regressive waves).

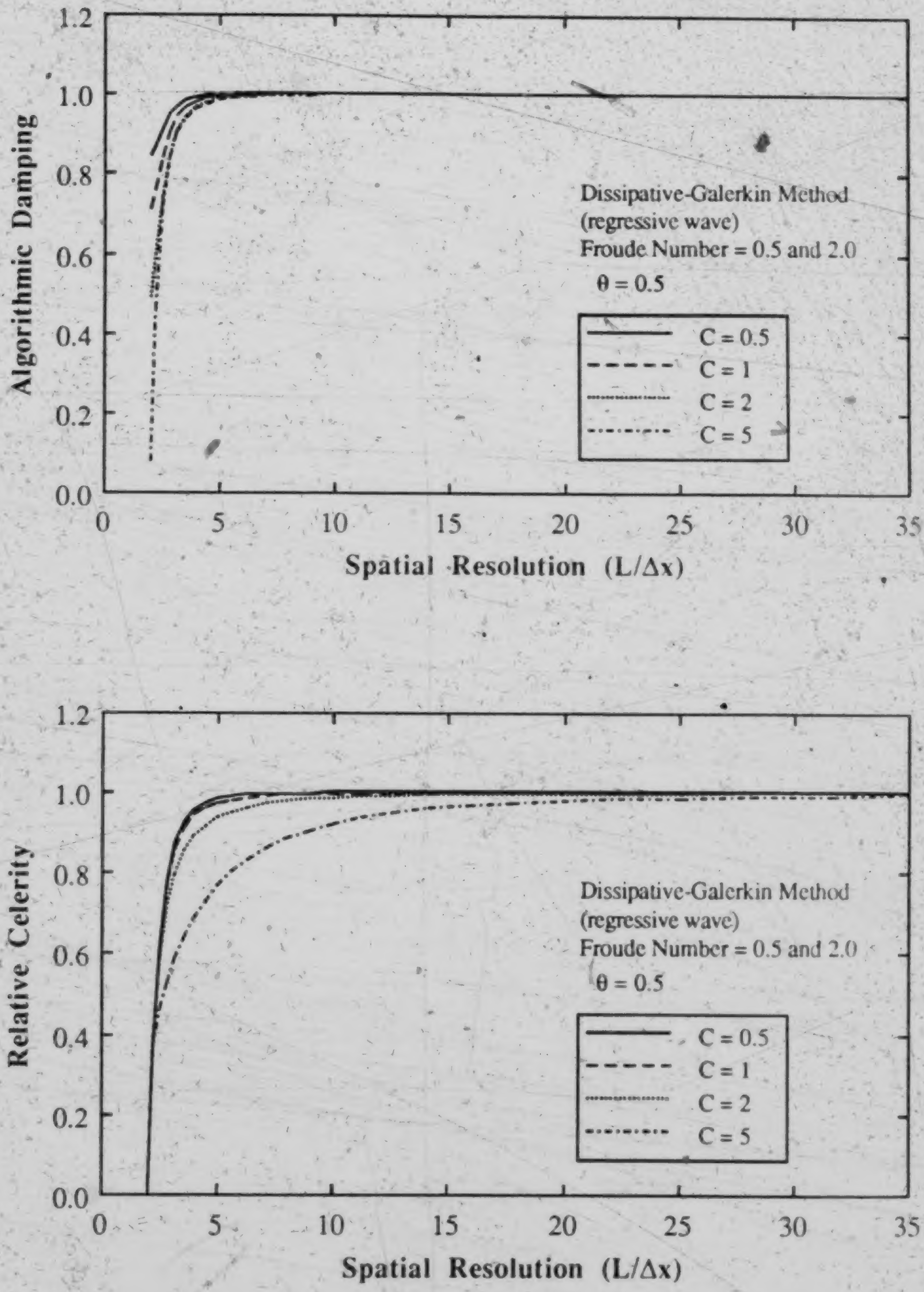
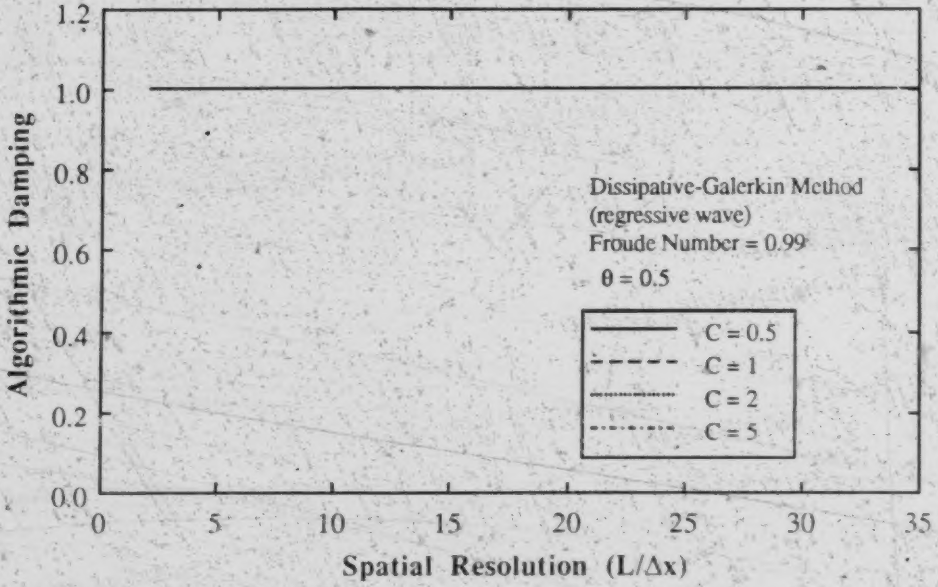


Figure 4.17 Results of stability analysis, Dissipative-Galerkin Method, **F** = 0.5 and Fr = 2.0, θ = 0.5 (regressive wave).



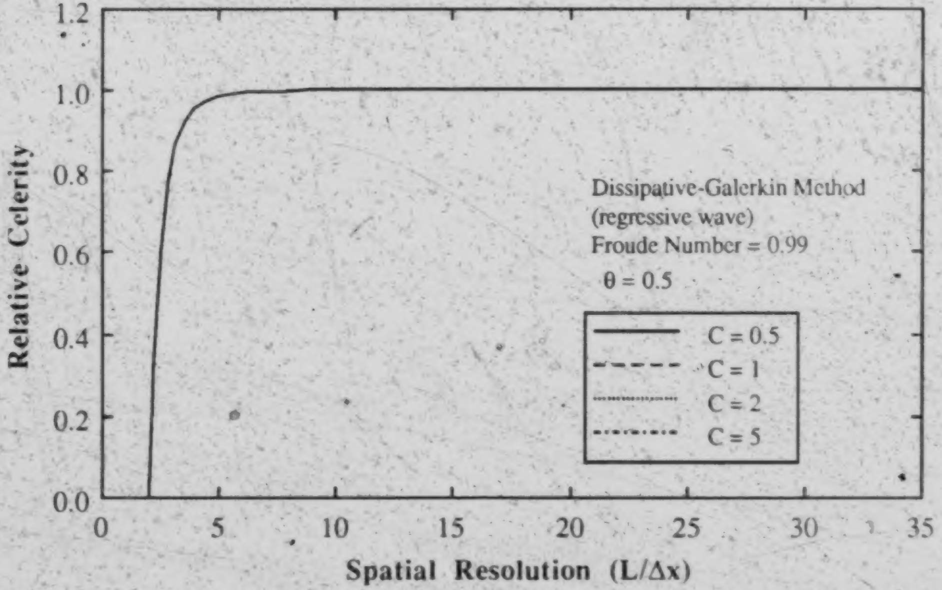


Figure 4.18 Results of stability analysis, Dissipative-Galerkin Method, **F** = 0.99, θ = 0.5 (regressive wave).

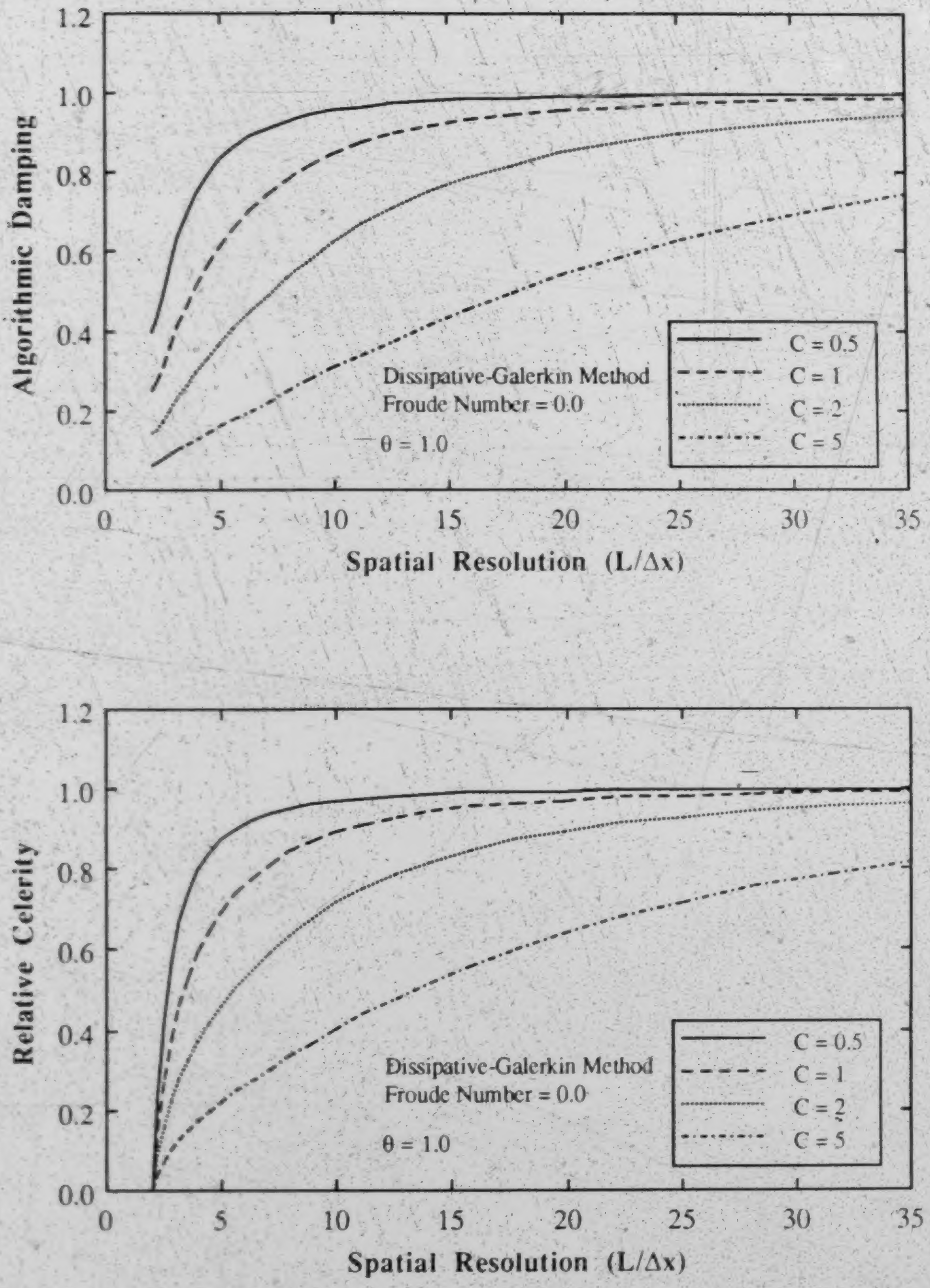
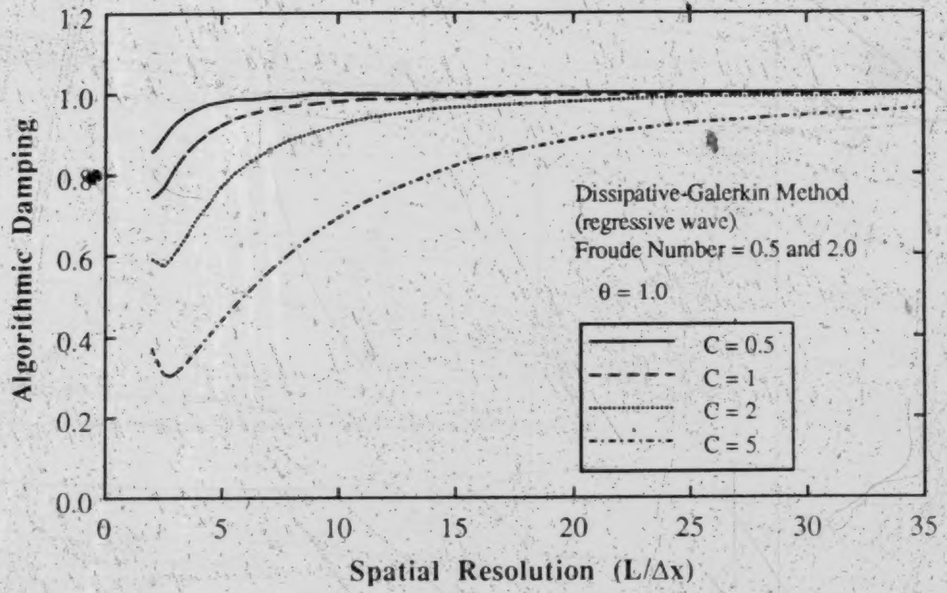


Figure 4.19 Results of stability analysis, Dissipative-Galerkin Method, F = 0.0, θ = 1.0 (progressive and regressive waves).



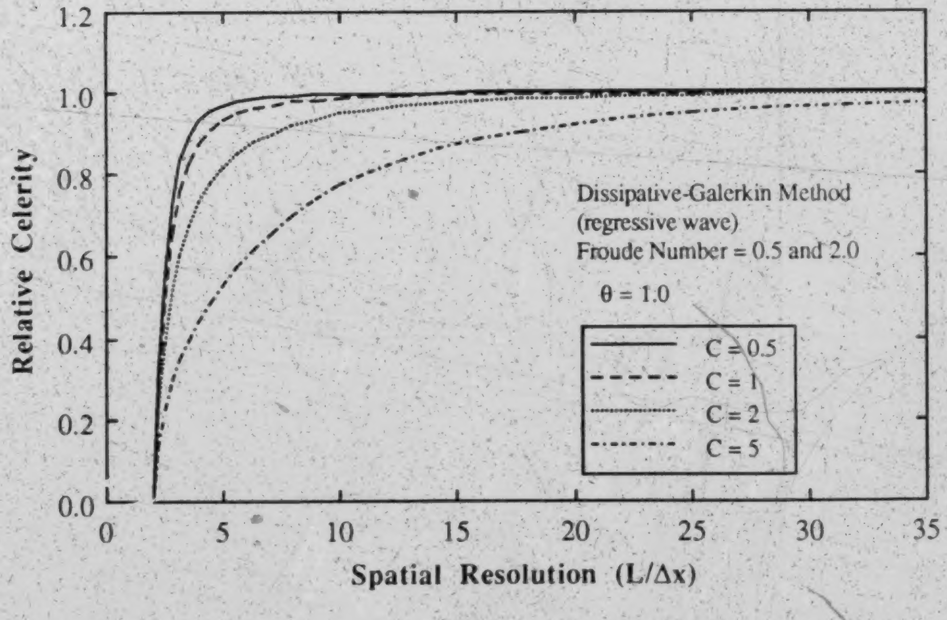


Figure 4.20 Results of stability analysis, Dissipative-Galerkin Method, F = 0.5 and 2.0, θ = 1.0 (regressive wave).

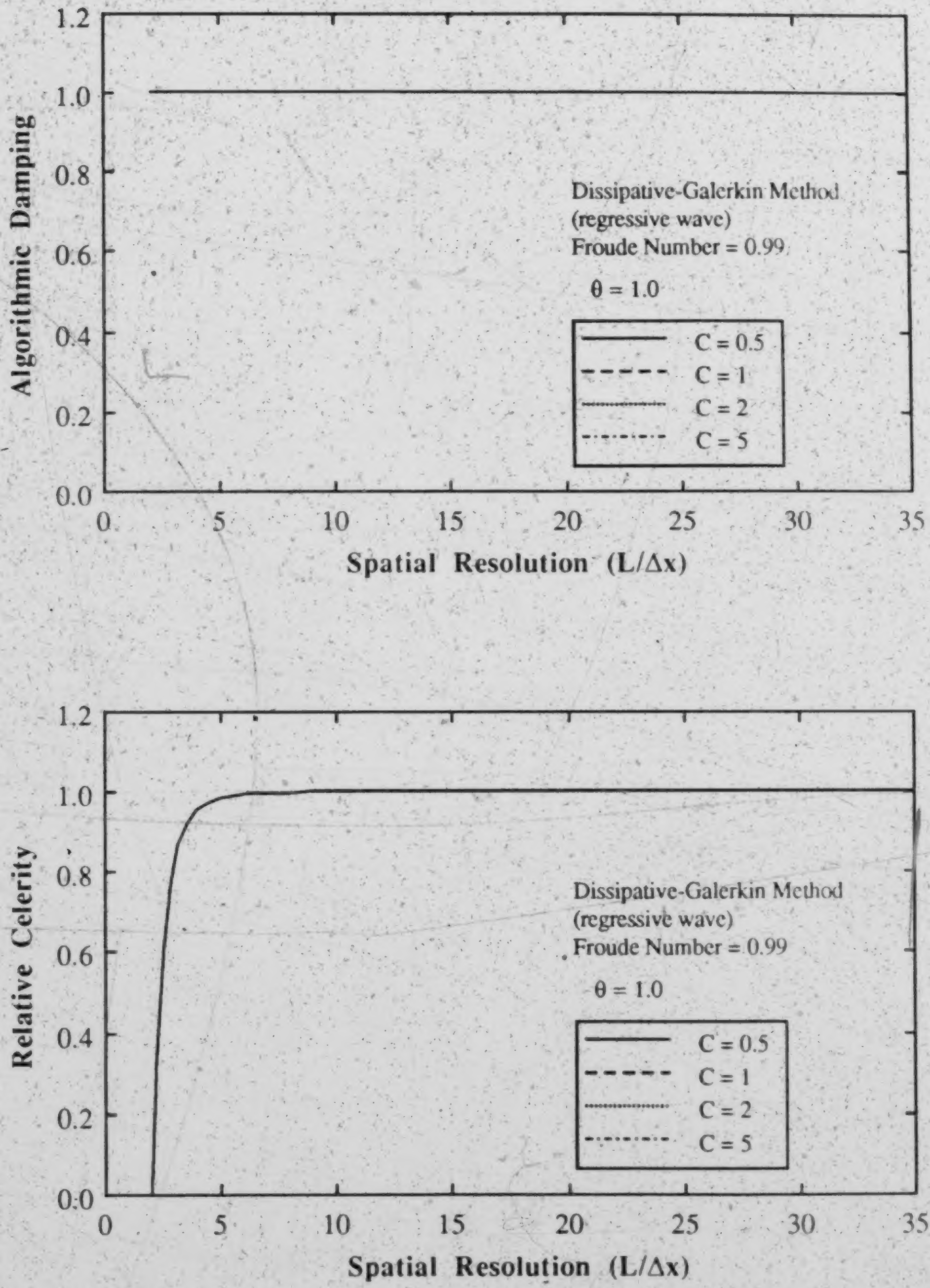
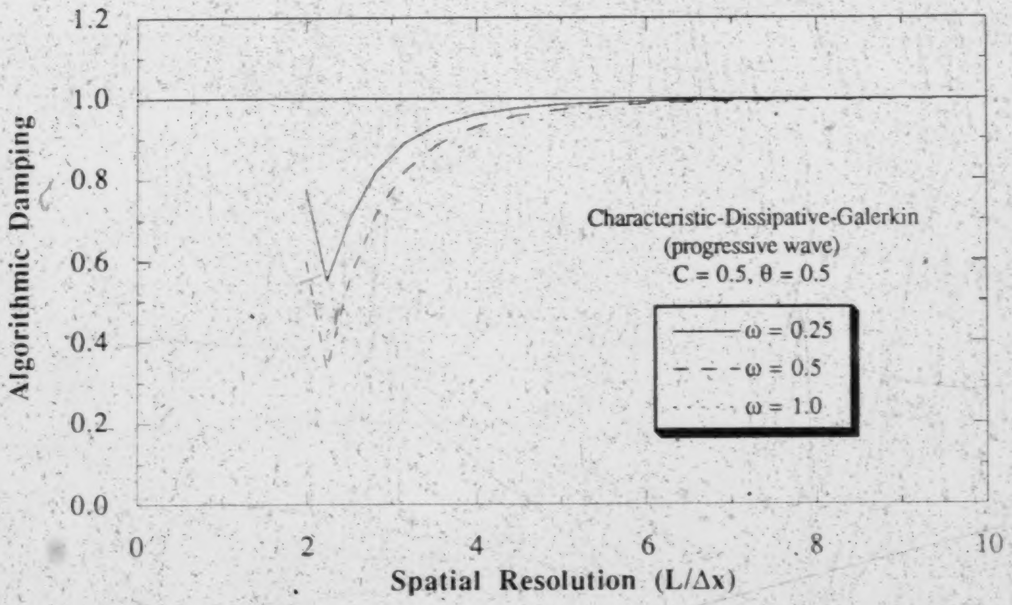


Figure 4.21. Results of stability analysis, Dissipative-Galerkin Method, **F** = 0.99, θ = 1.0 (regressive wave).



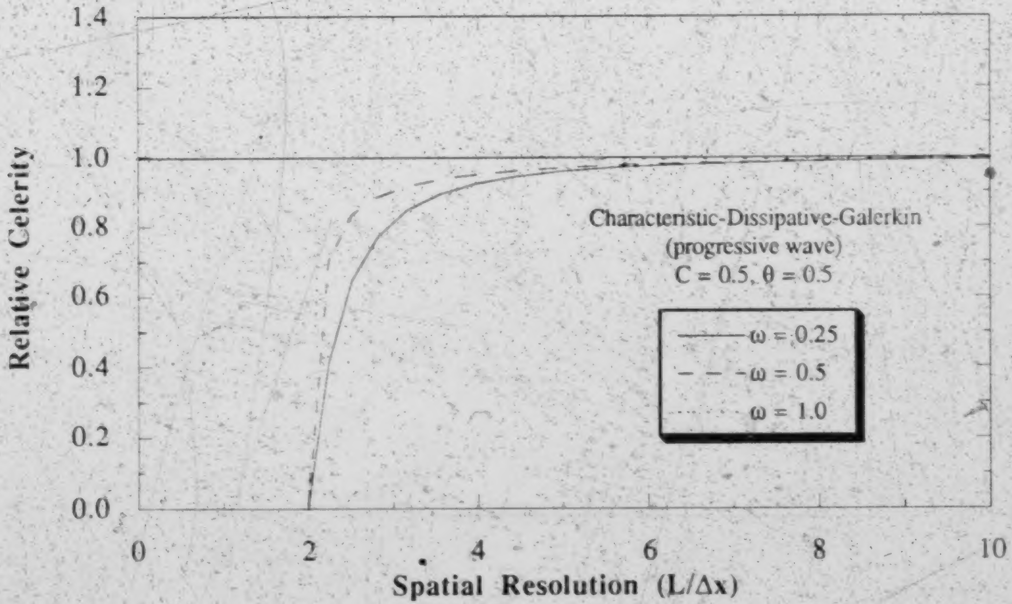
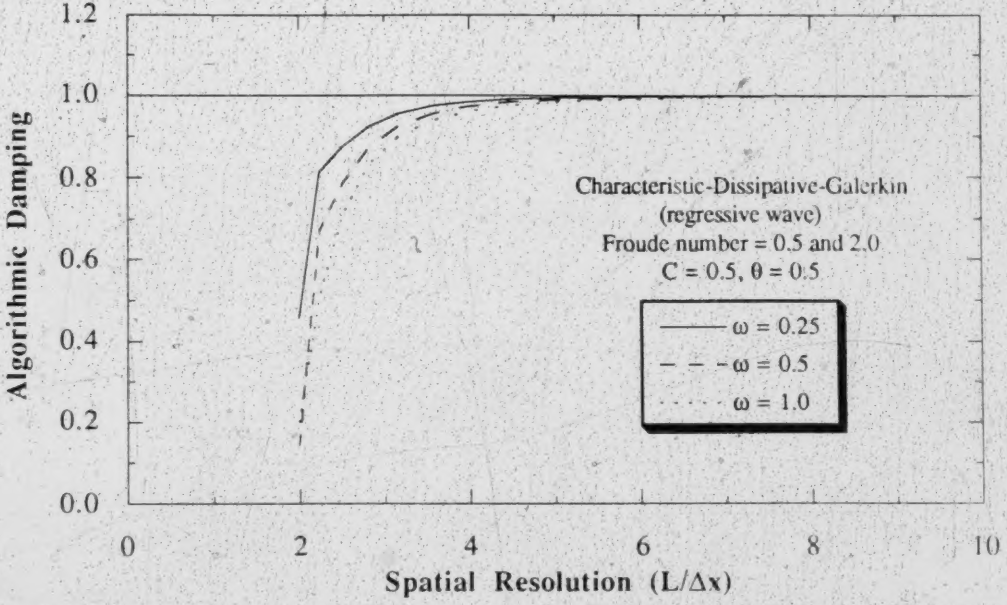


Figure 4.22 Results of stability analysis for ω = 0.25, 0.5 and 1.0, Characteristic-Dissipative-Galerkin Method, C = 0.5, θ = 0.5 (progressive wave).



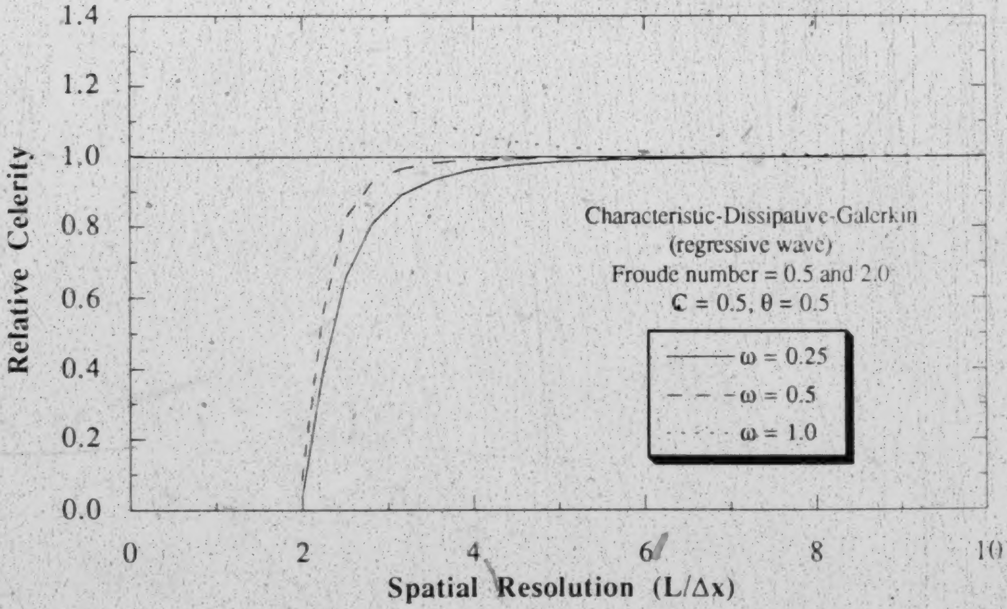
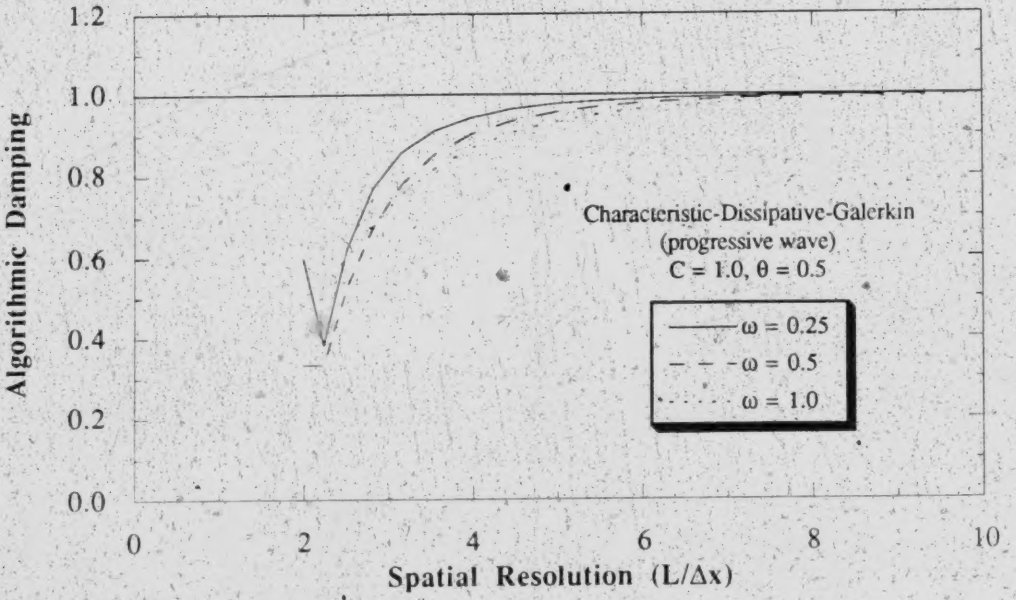


Figure 4.23 Results of stability analysis for ω = 0.25, 0.5 and 1.0, Characteristic-Dissipative-Galerkin Method, **F** = 0.5 and 2.0, C = 0.5, θ = 0.5 (regressive wave).



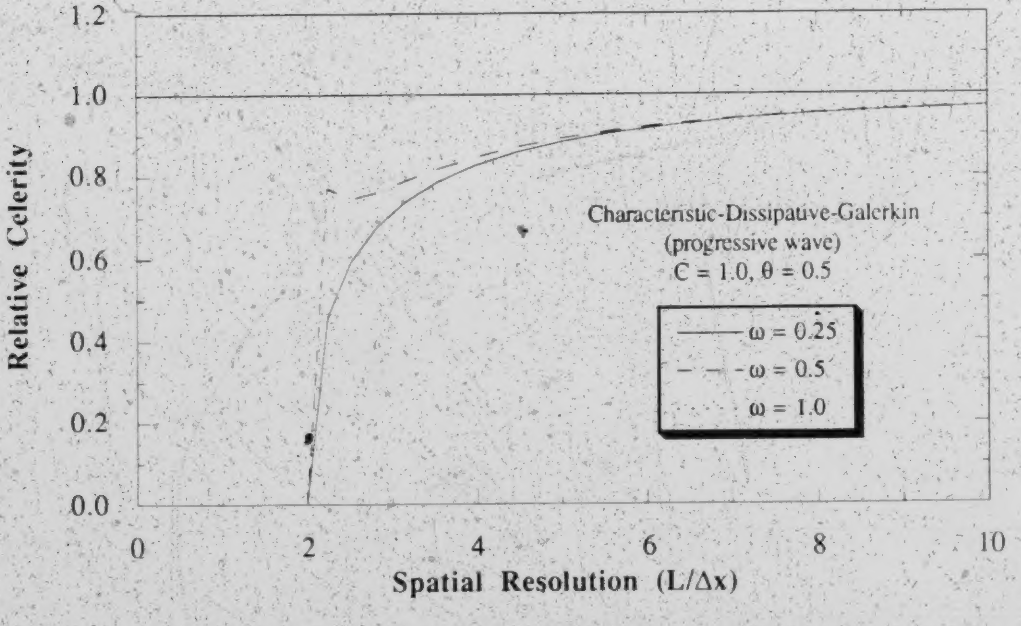
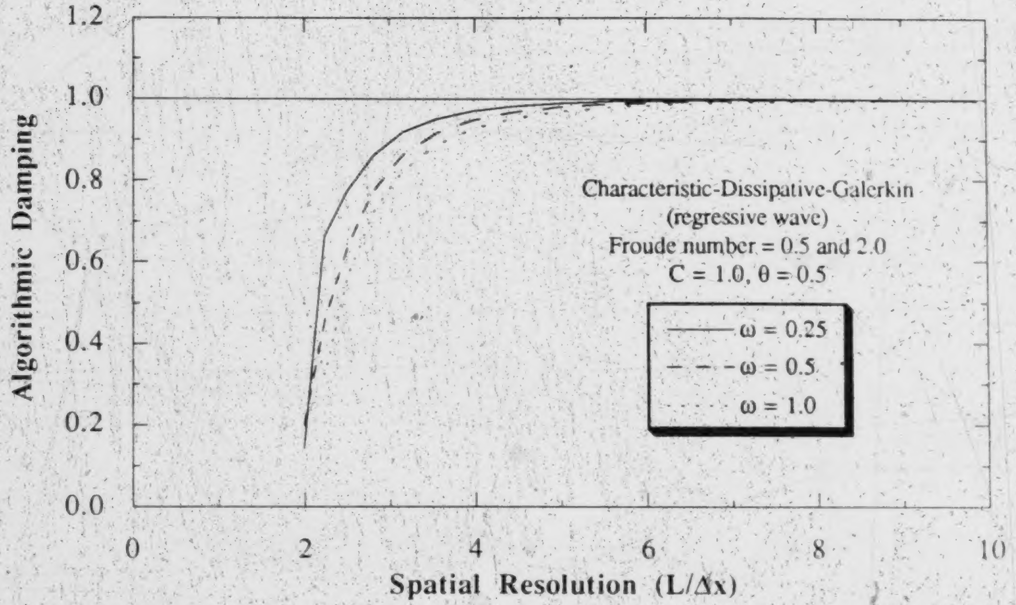


Figure 4.24 Results of stability analysis for ω = 0.25, 0.5 and 1.0, Characteristic-Dissipative-Galerkin Method, C = 1.0, θ = 0.5 (progressive wave).



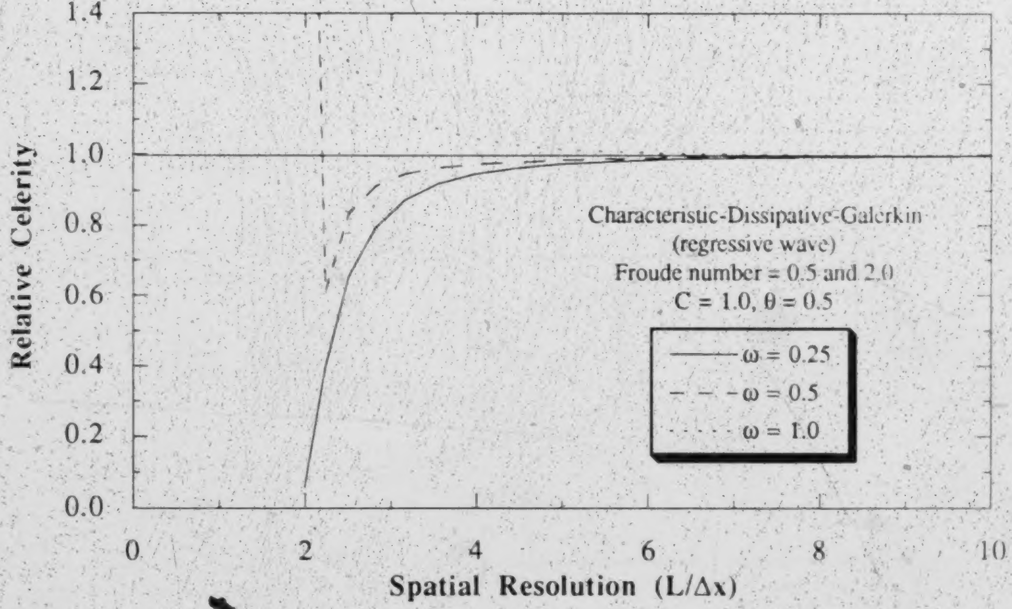
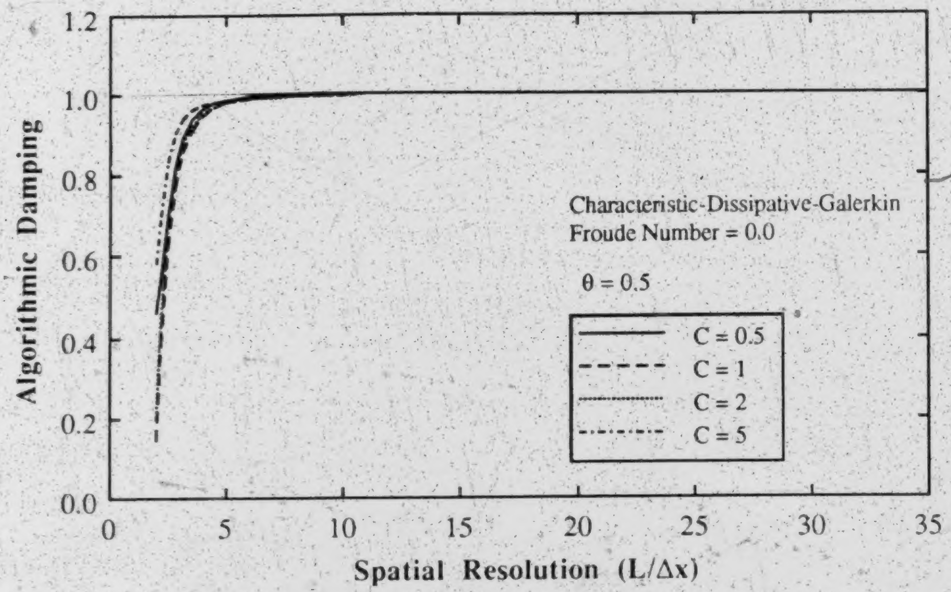


Figure 4.25 Results of stability analysis for ω = 0.25, 0.5 and 1.0, Characteristic-Dissipative-Galerkin Method, F = 0.5 and 2.0, C = 1.0, θ = 0.5 (regressive wave).



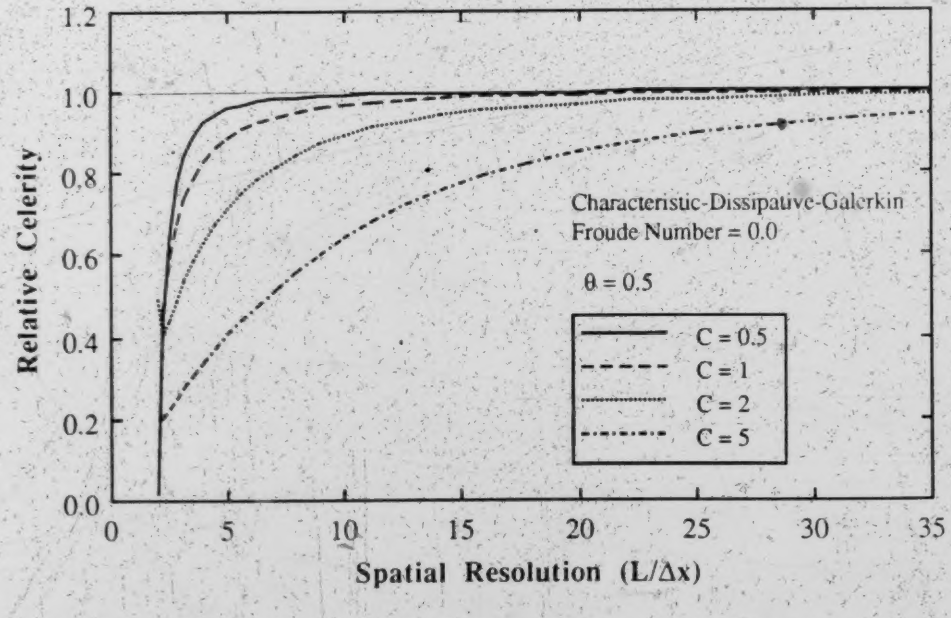
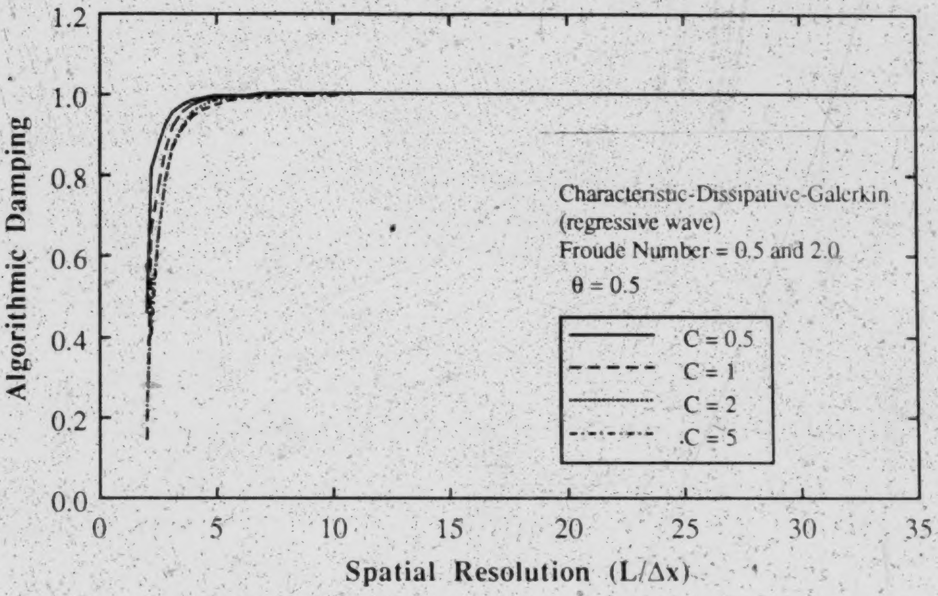


Figure 4.26 Results of stability analysis, Characteristic-Dissipative-Galerkin Method, **F** = 0.0, θ = 0.5 (progressive and regressive waves).



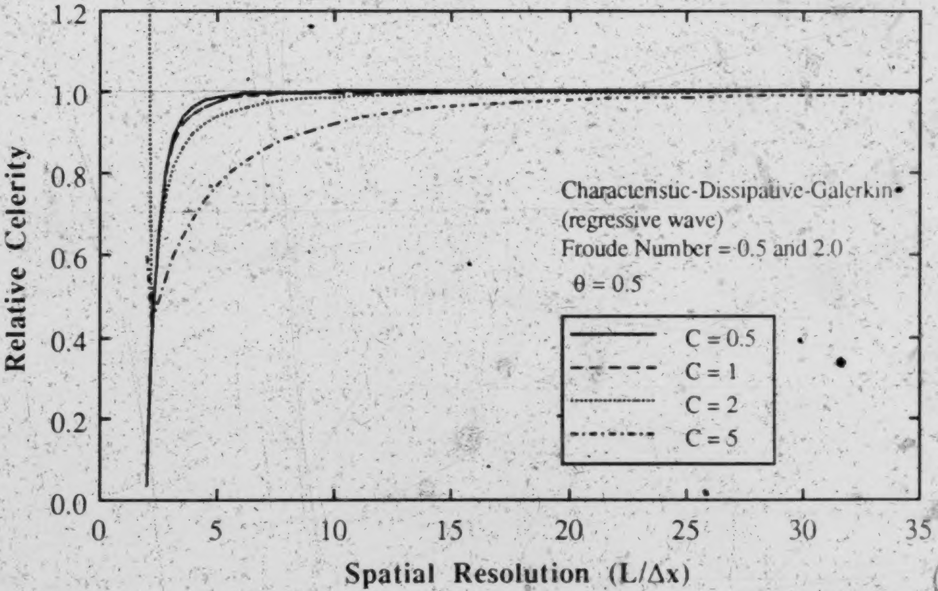
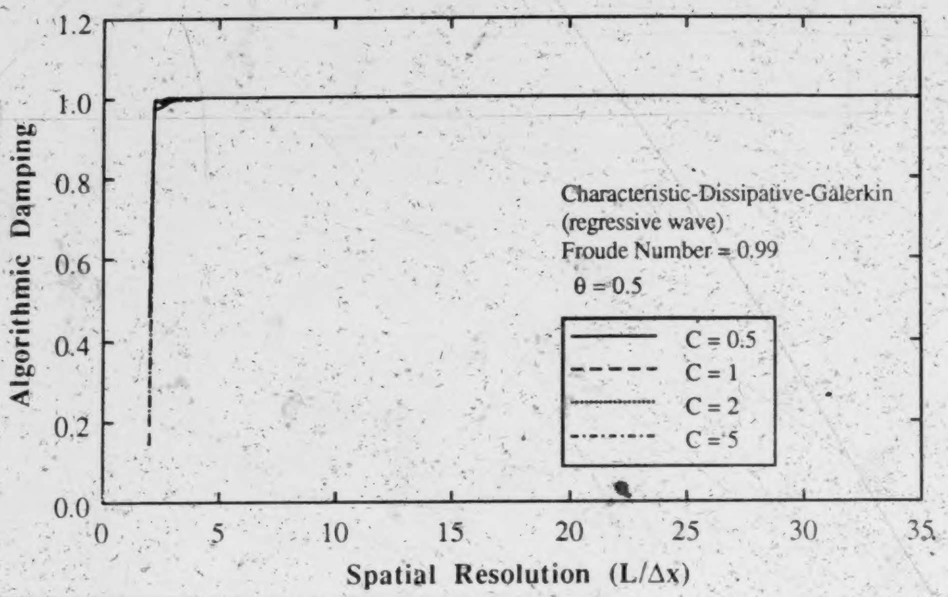


Figure 4.27 Results of stability analysis, Characteristic-Dissipative-Galerkin Method, **F** = 0.5 and 2.0, θ = 0.5 (regressive wave).



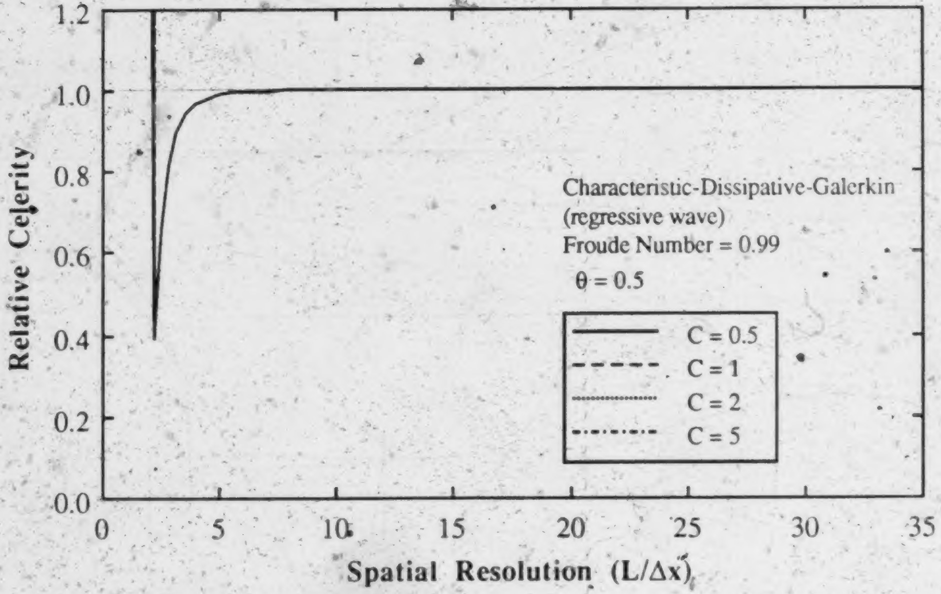
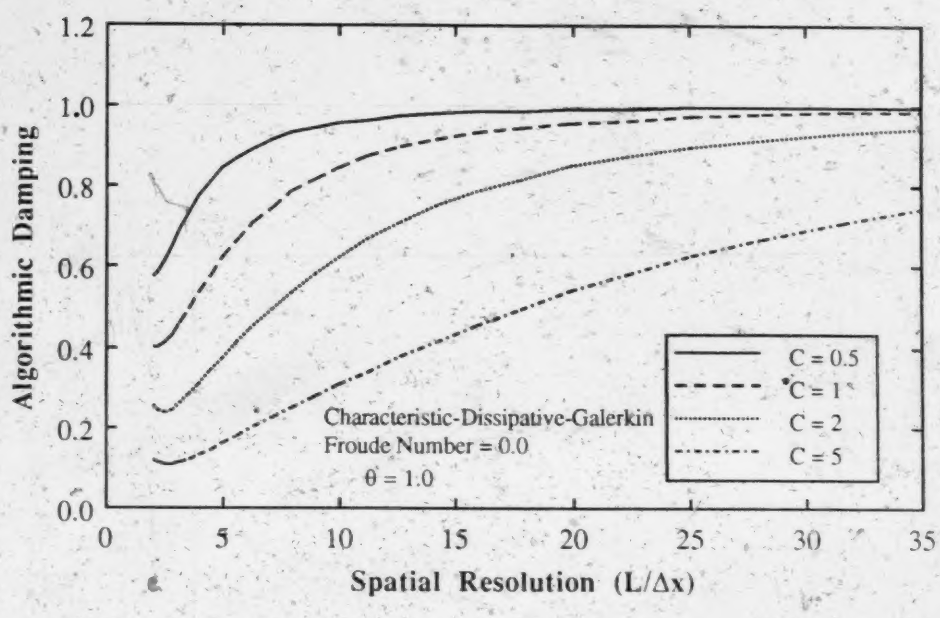


Figure 4.28 Results of stability analysis, Characteristic-Dissipative-Galerkin Method, $F = 0.99$, $\theta = 0.5$ (regressive wave).



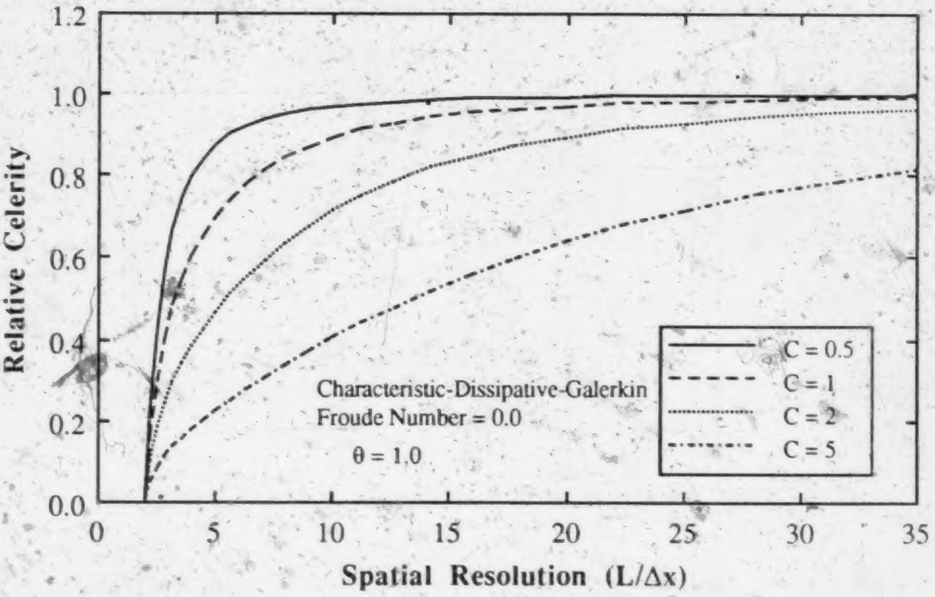
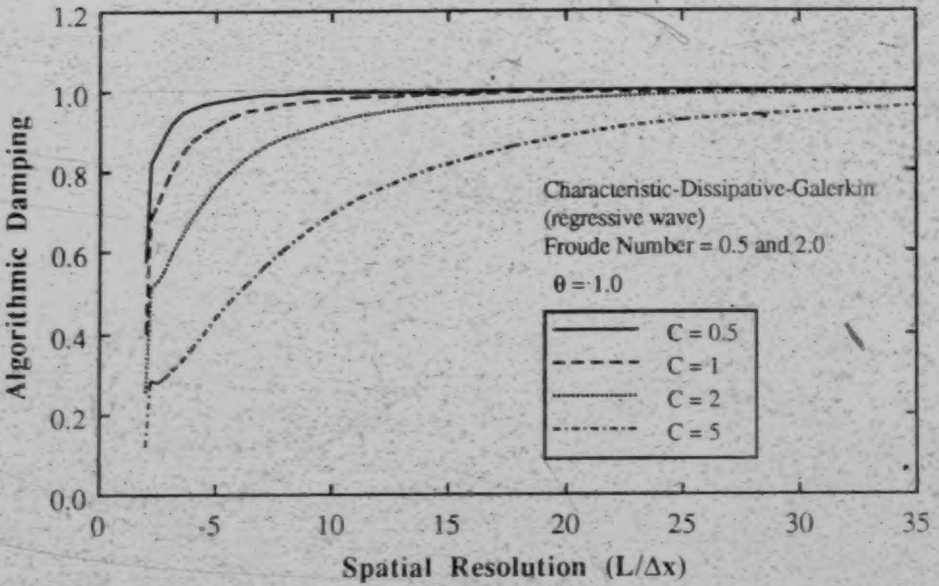


Figure 4.29 Results of stability analysis, Characteristic-Dissipative-Galerkin Method, **F** = 0.0, θ = 1.0 (progressive and regressive waves).



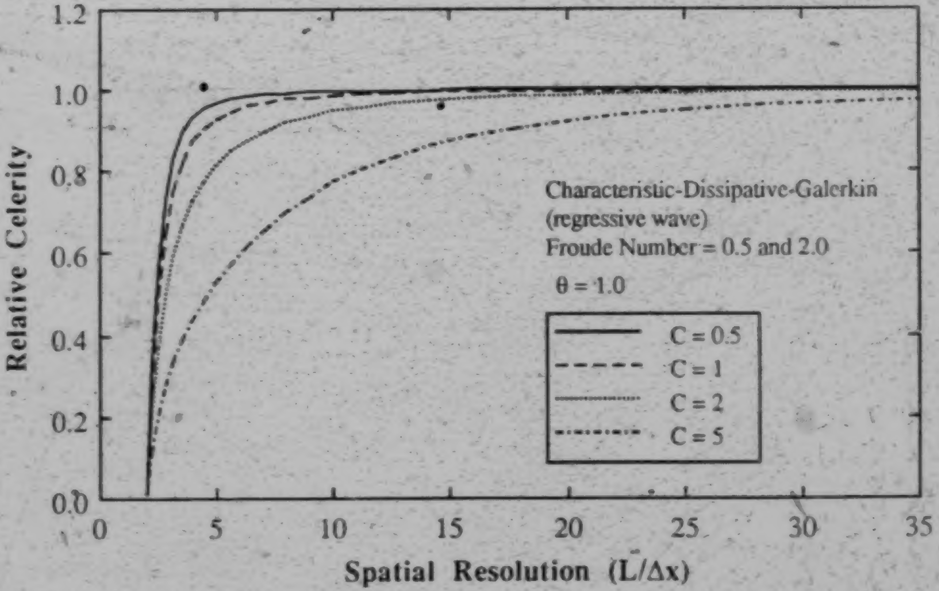
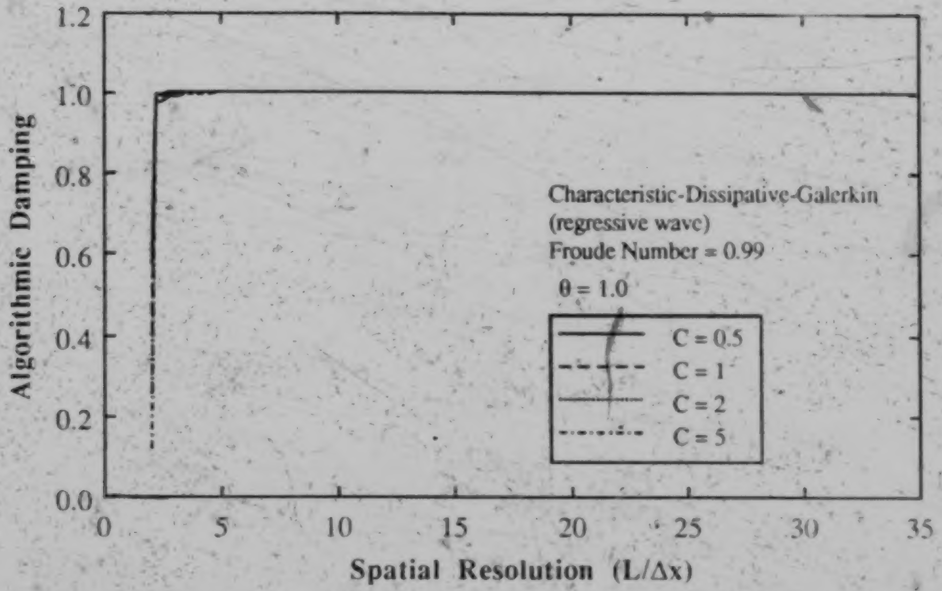


Figure 4.30 Results of stability analysis, Characteristic-Dissipative-Galerkin Method, F = 0.5 and 2.0, θ = 1.0 (regressive wave).



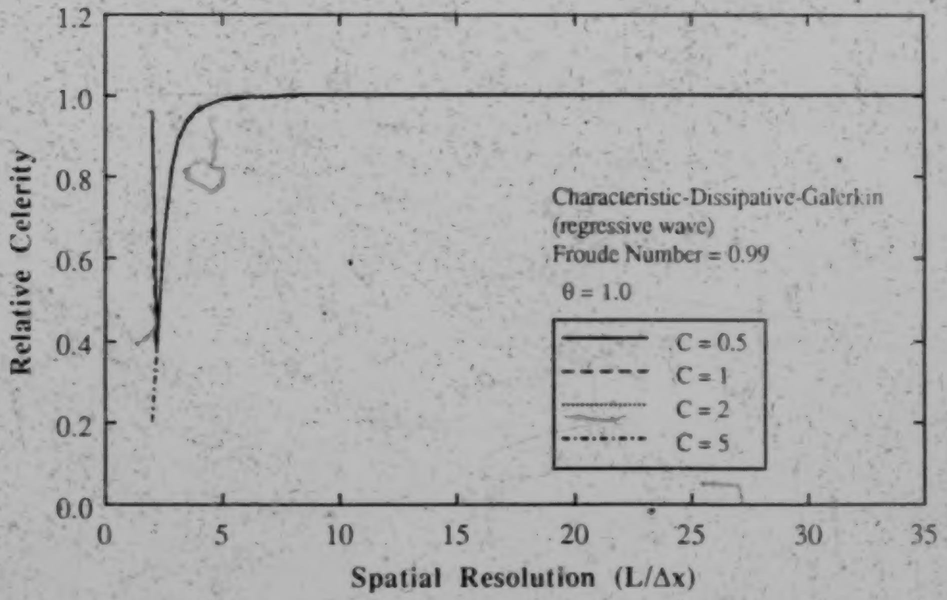
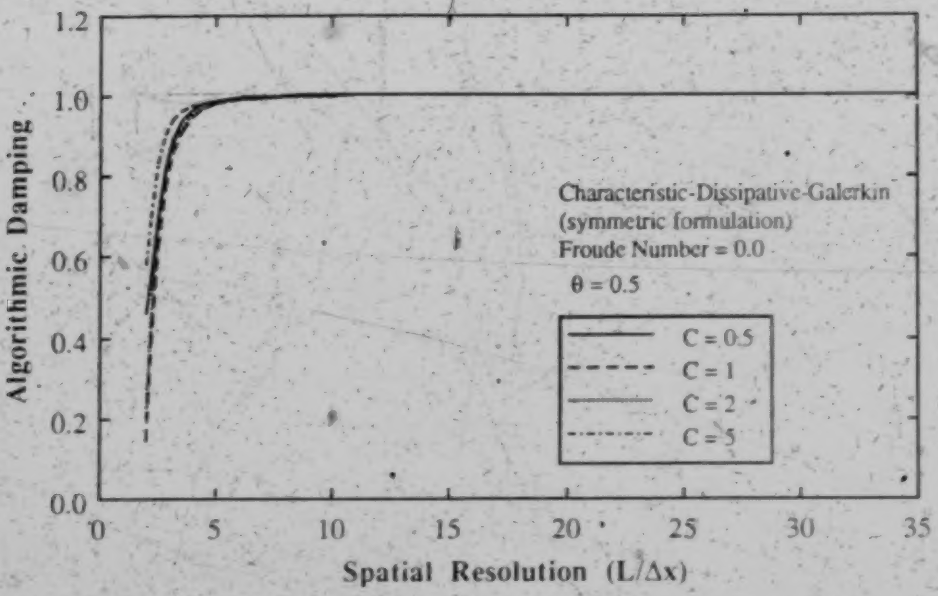


Figure 4.31 Results of stability analysis, Characteristic-Dissipative-Galerkin Method, **F** = 0.99, θ = 1.0 (regressive wave).



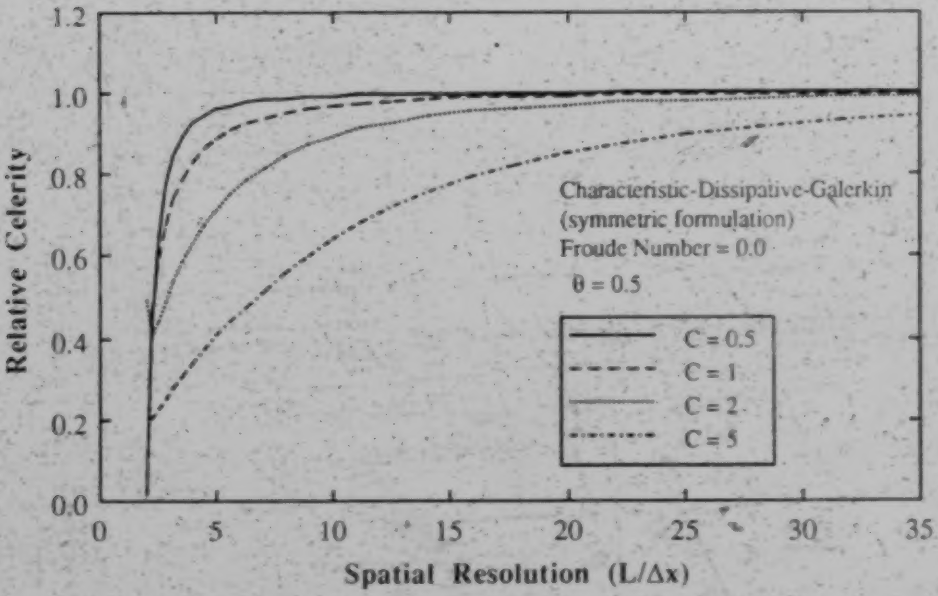


Figure 4.32 Results of stability analysis, Characteristic-Dissipative-Galerkin Method, F = 0.0, θ = 0.5 (symmetric formulation).

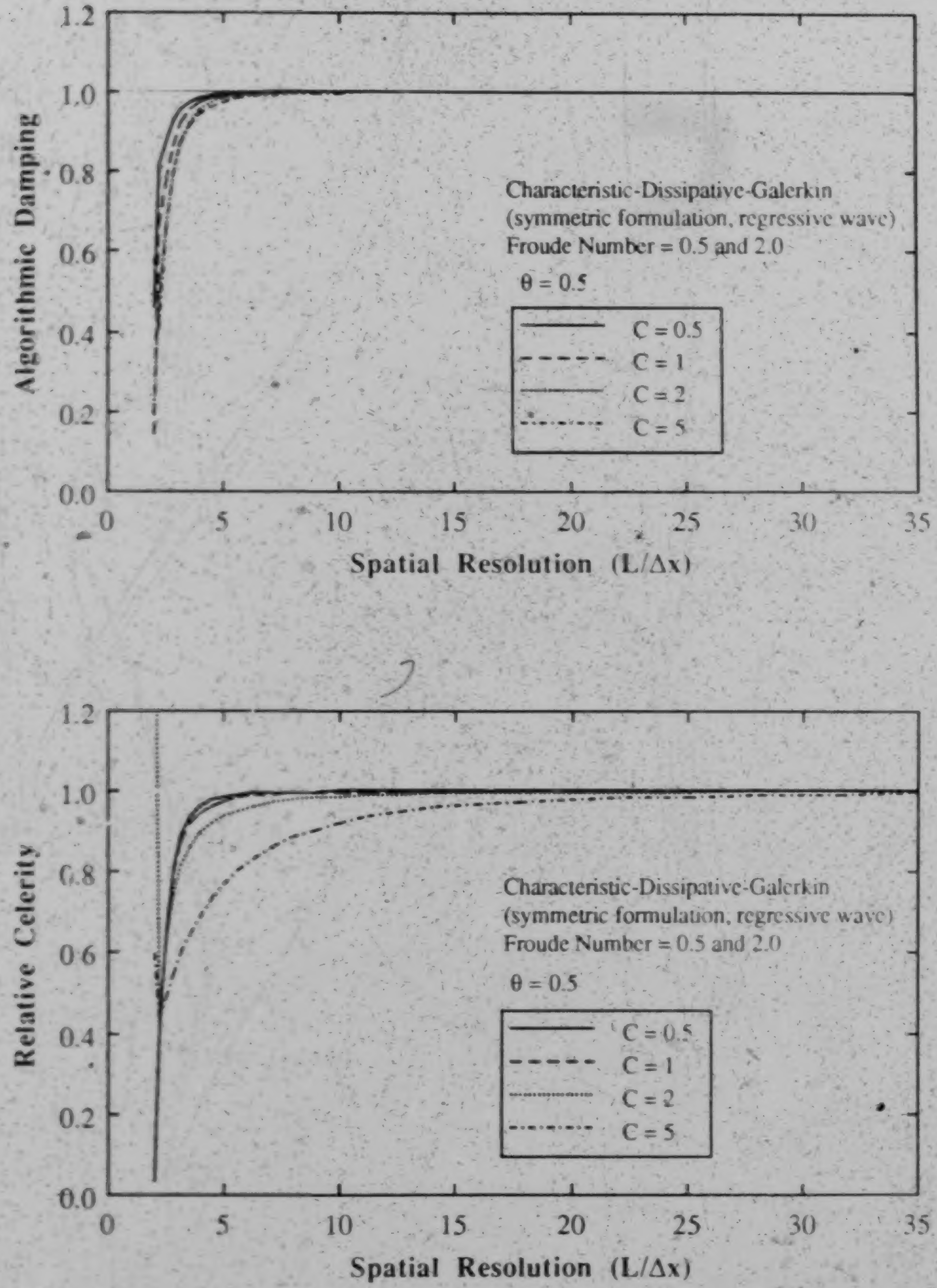
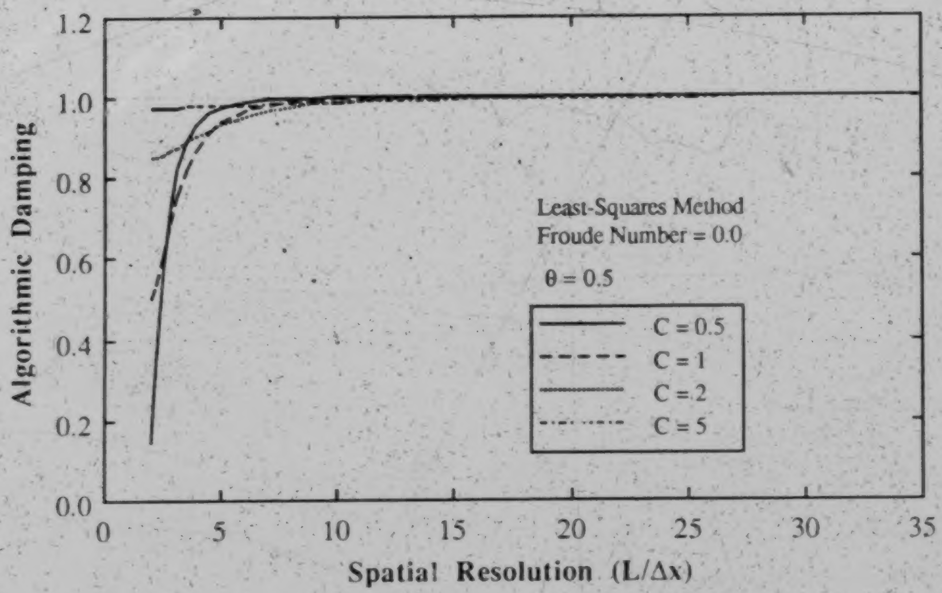


Figure 4.33 Results of stability analysis, Characteristic-Dissipative-Galerkin Method, F = 0.5 and 2.0, θ = 0.5 (symmetric formulation, regressive wave).



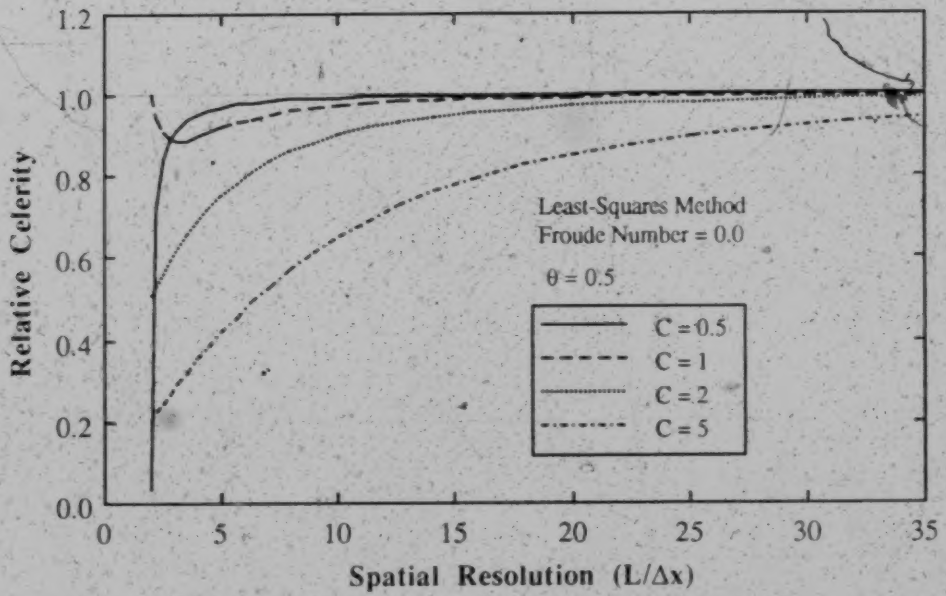


Figure 4.34 Results of stability analysis, Least-Squares Method, **F** = 0.0, θ = 0.5 (progressive and regressive waves).

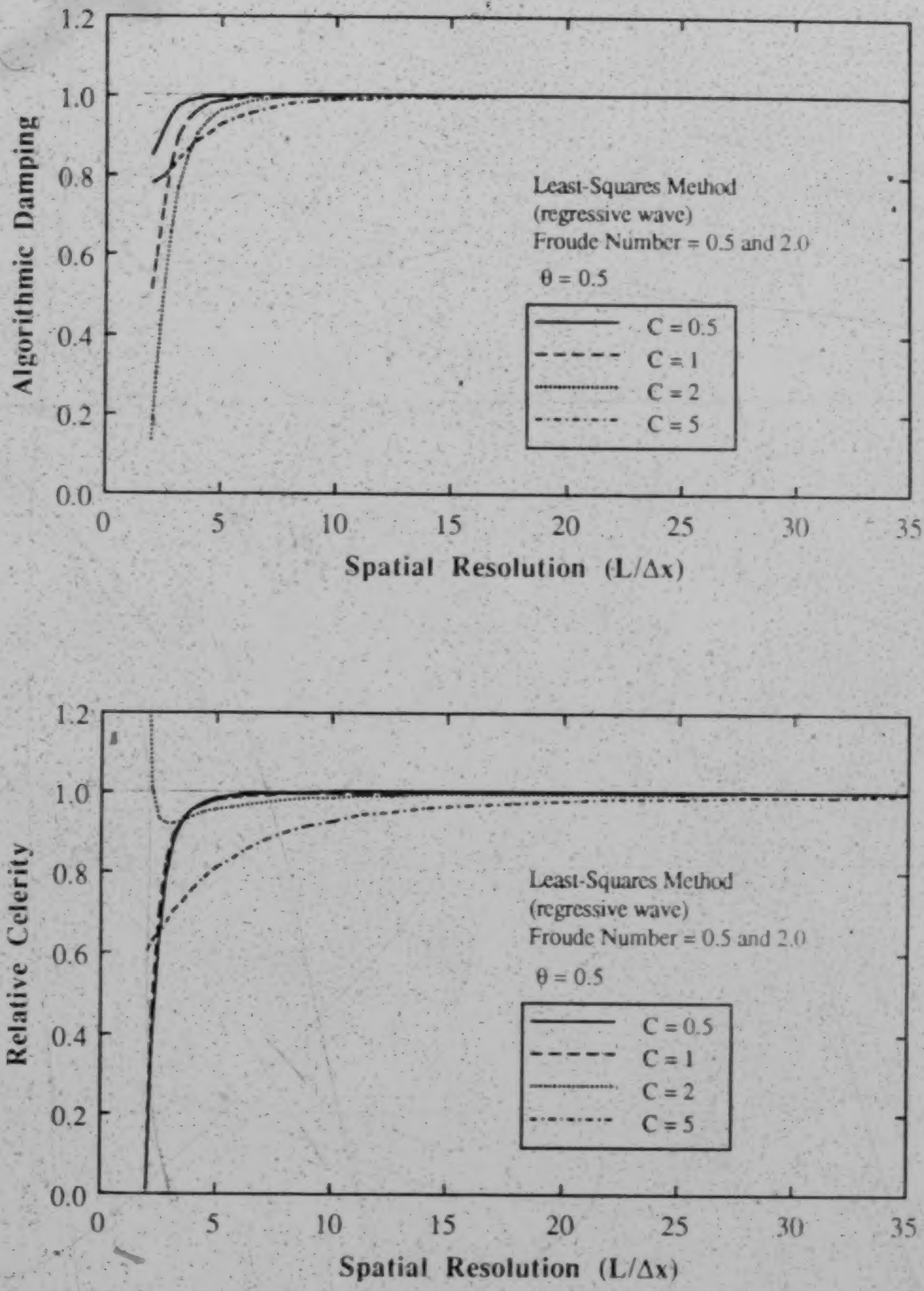
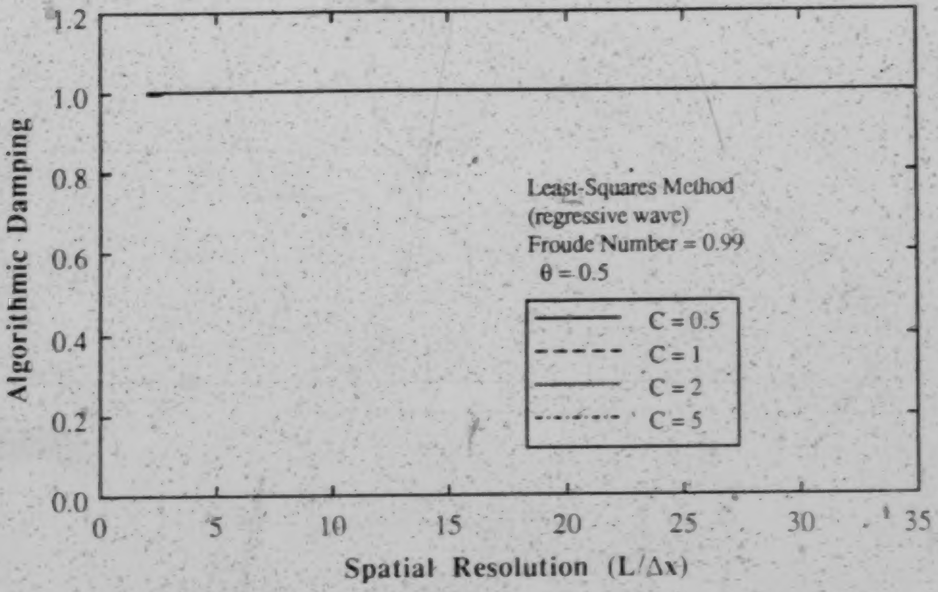


Figure 4.35 Results of stability analysis, Least-Squares Method, F = 0.5 and 2.0, θ = 0.5 (regressive wave).



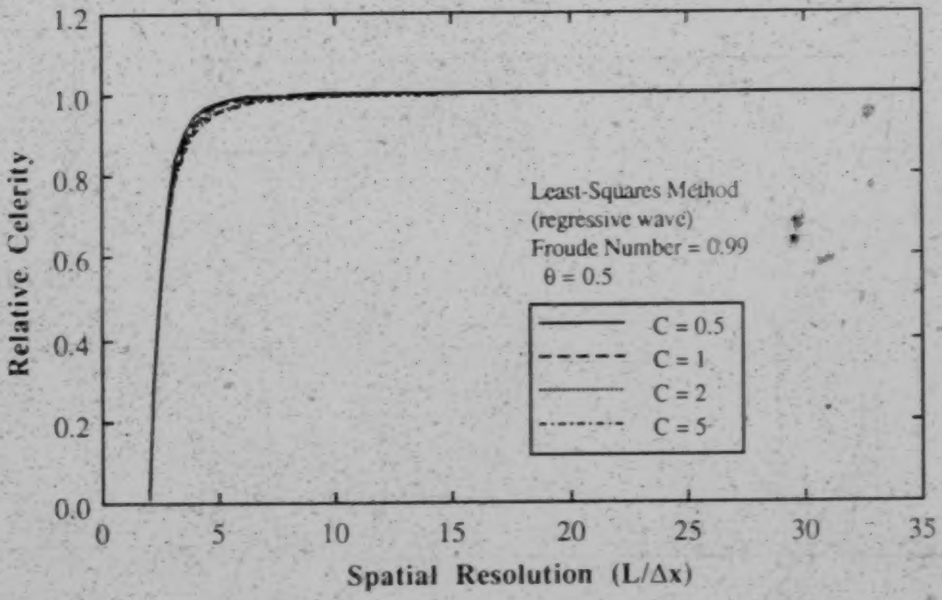
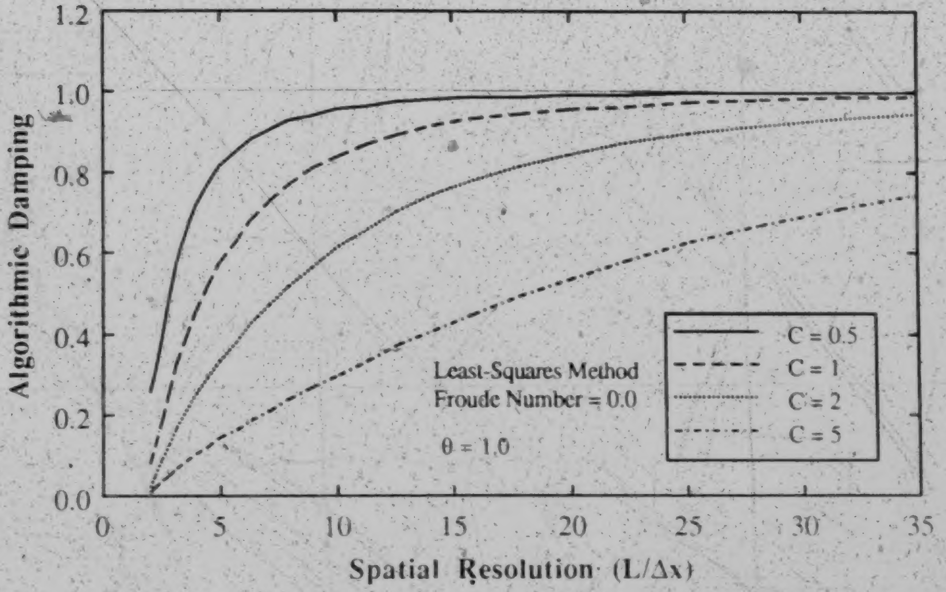


Figure 4.36 Results of stability analysis, Least-Squares Method, F = 0.99, θ = 0.5 (regressive wave).



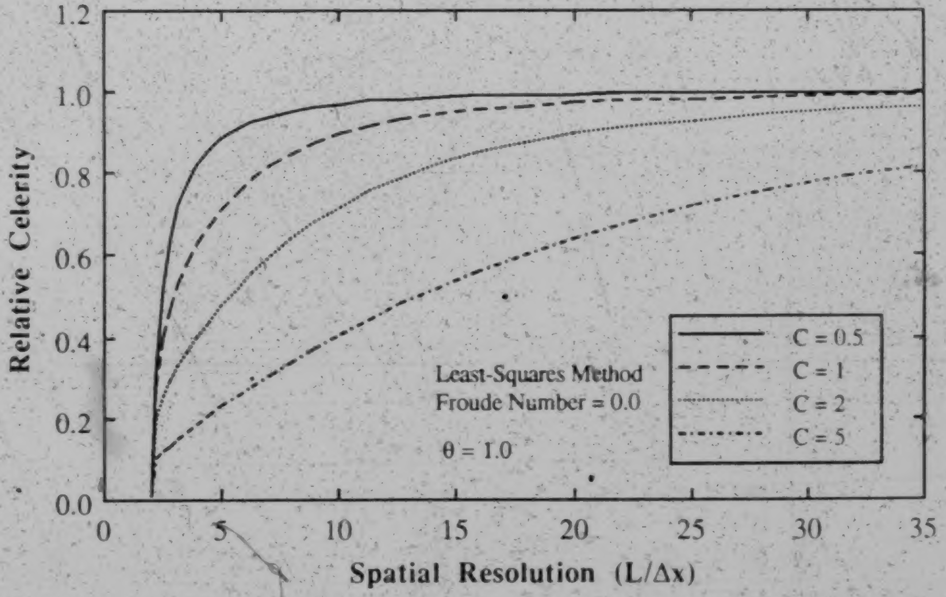
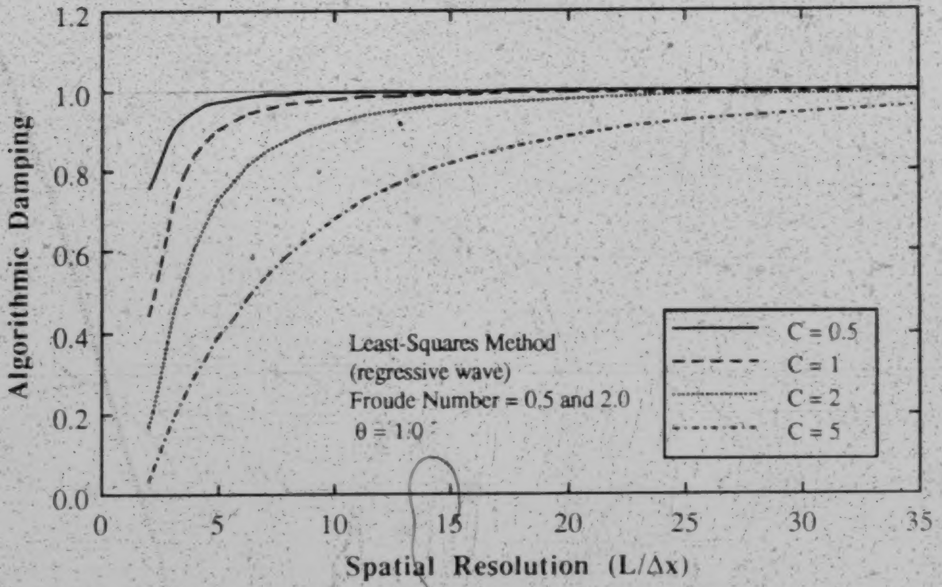


Figure 4.37 Results of stability analysis, Least-Squares Method, **F** = 0.0, θ = 1.0 (progressive and regressive waves).



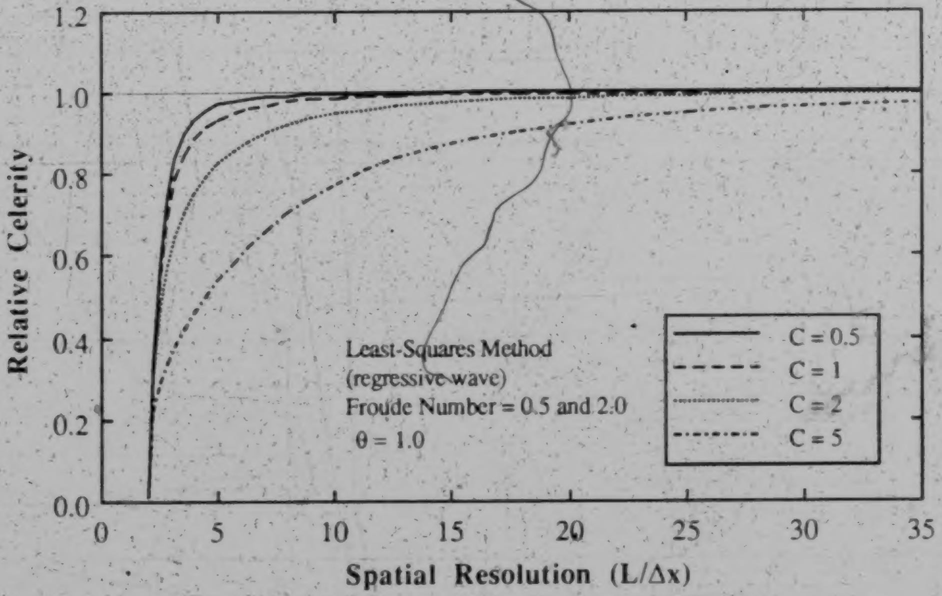


Figure 4.38 Results of stability analysis, Least-Squares Method, **F** = 0.5 and 2.0, θ = 1.0 (regressive wave).

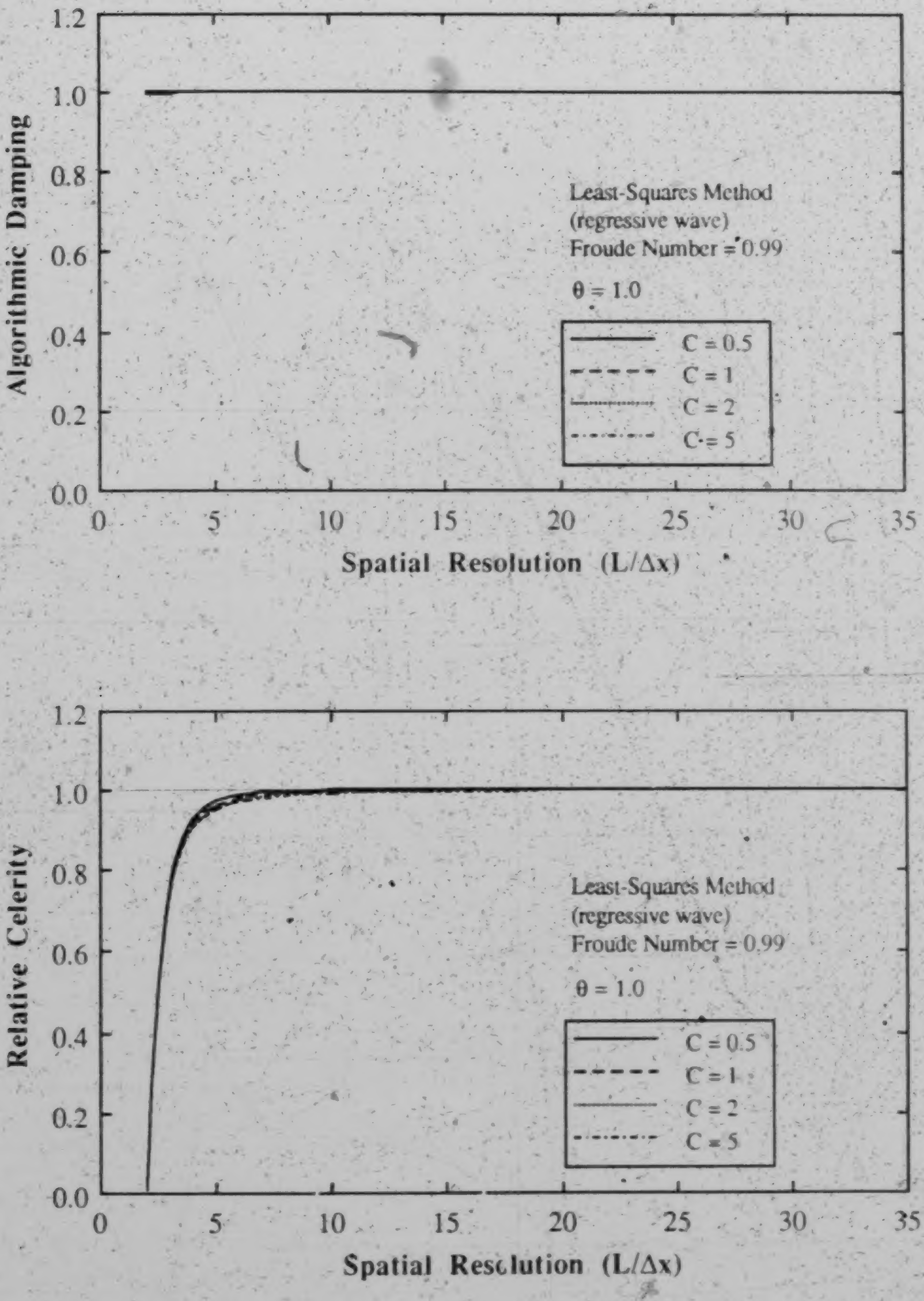
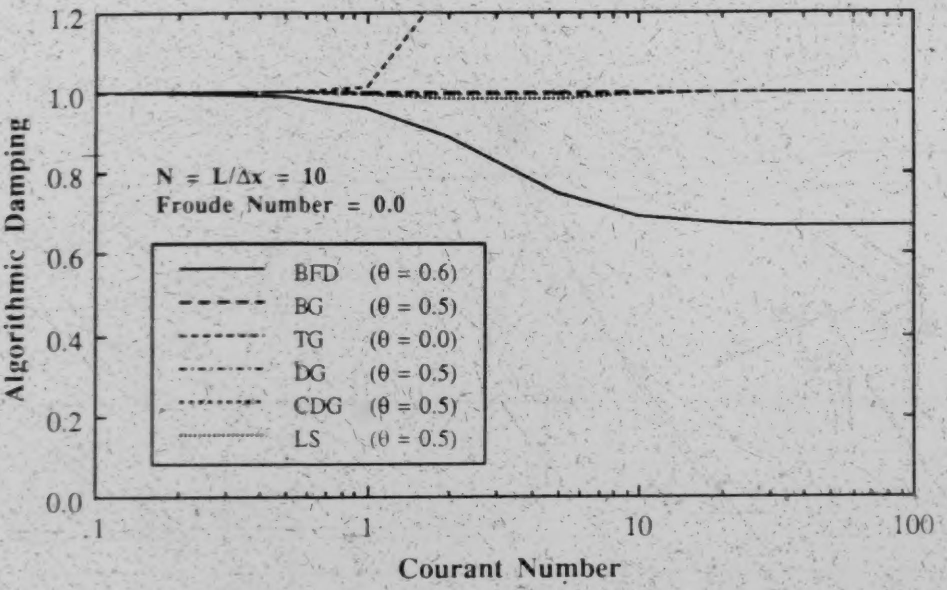


Figure 4.39 Results of stability analysis, Least-Squares Method, **F** = 0.99, θ = 1.0 (regressive wave).



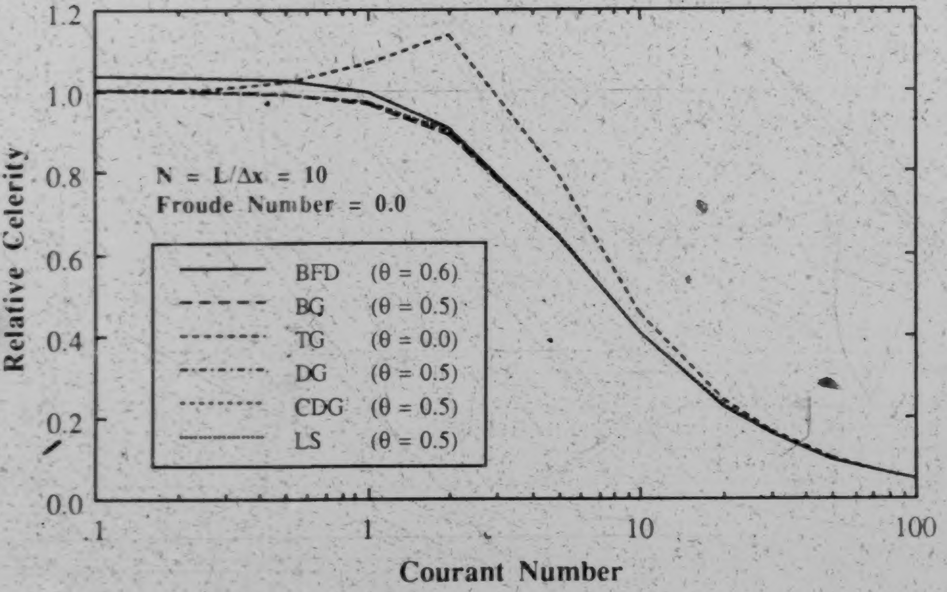
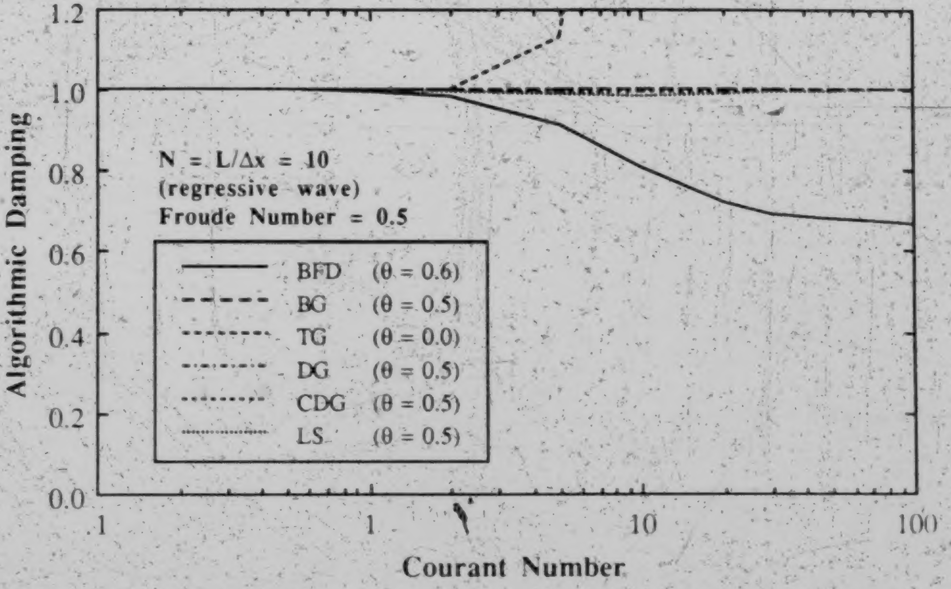


Figure 4.40 Comparison of results of stability analysis (progressive and regressive waves), N = L/Δx = 10, **F** = 0.0, θ = 0.5 for all methods except Box Finite Difference (θ = 0.6) and Taylor-Galerkin (θ = 0.0).



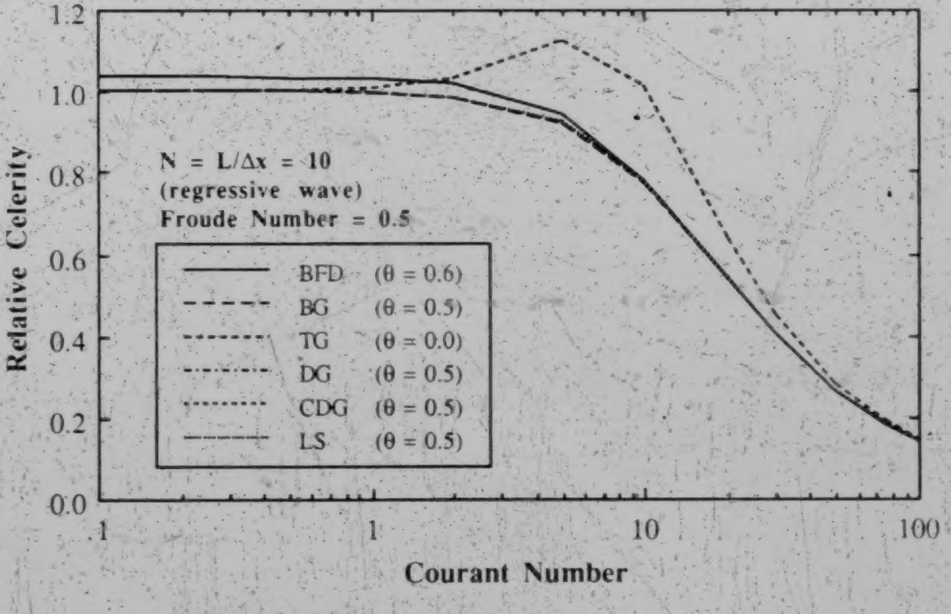


Figure 4.41 Comparison of results of stability analysis (regressive wave), N = L/Δx = 10, **F** = 0.5, θ = 0.5 for all methods except Box Finite Difference (θ = 0.6) and Taylor-Galerkin (θ = 0.0).

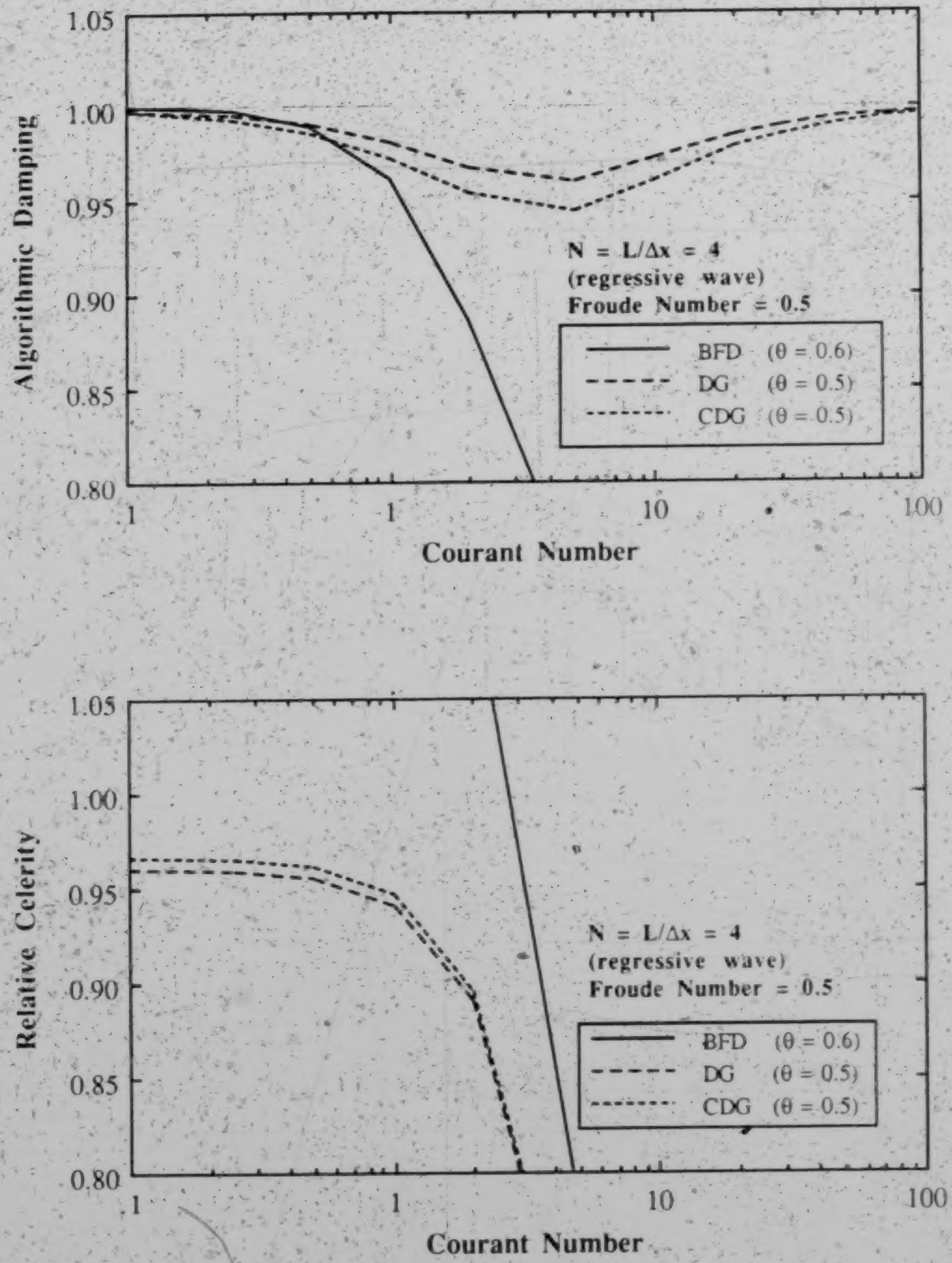
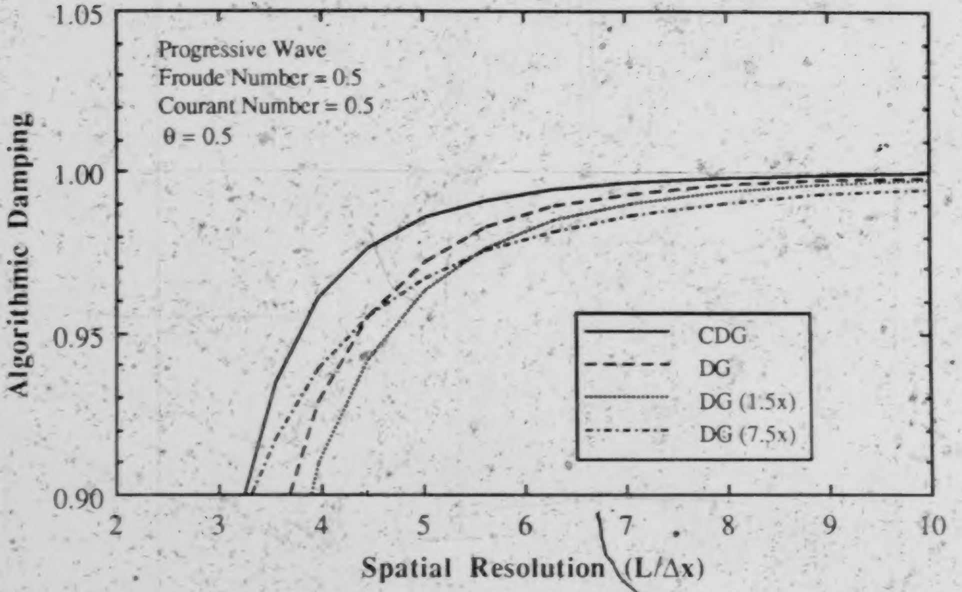


Figure 4.42 Comparison of results of stability analysis, N = L/Δx = 4, F = 0.5, θ = 0.5 (regressive wave).



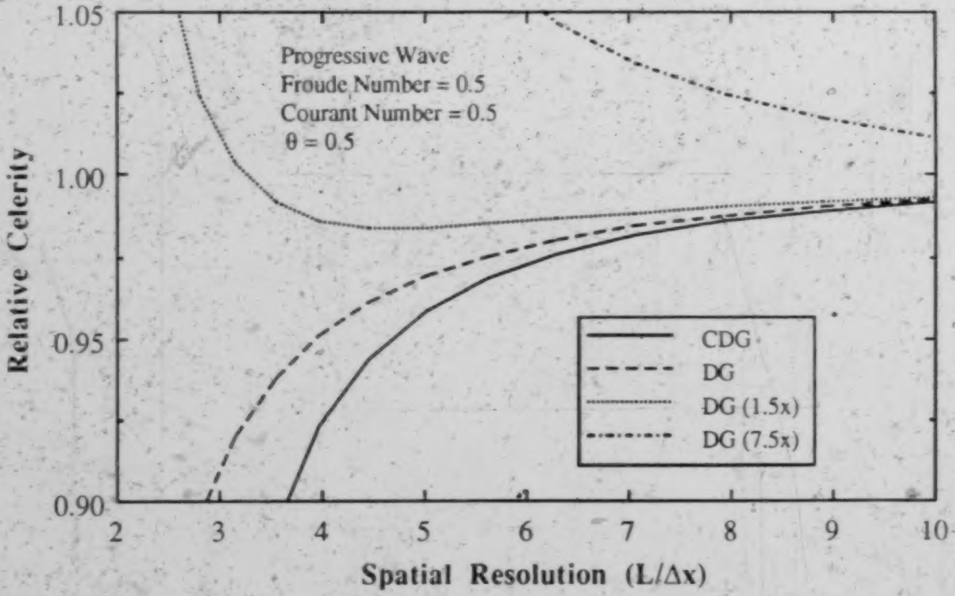


Figure 4.43 Effect of increasing the upwinding coefficient, ω, for the Dissipative-Galerkin Method, on the results of stability analysis, C = 0.5, F = 0.5, θ = 0.5 (progressive wave).

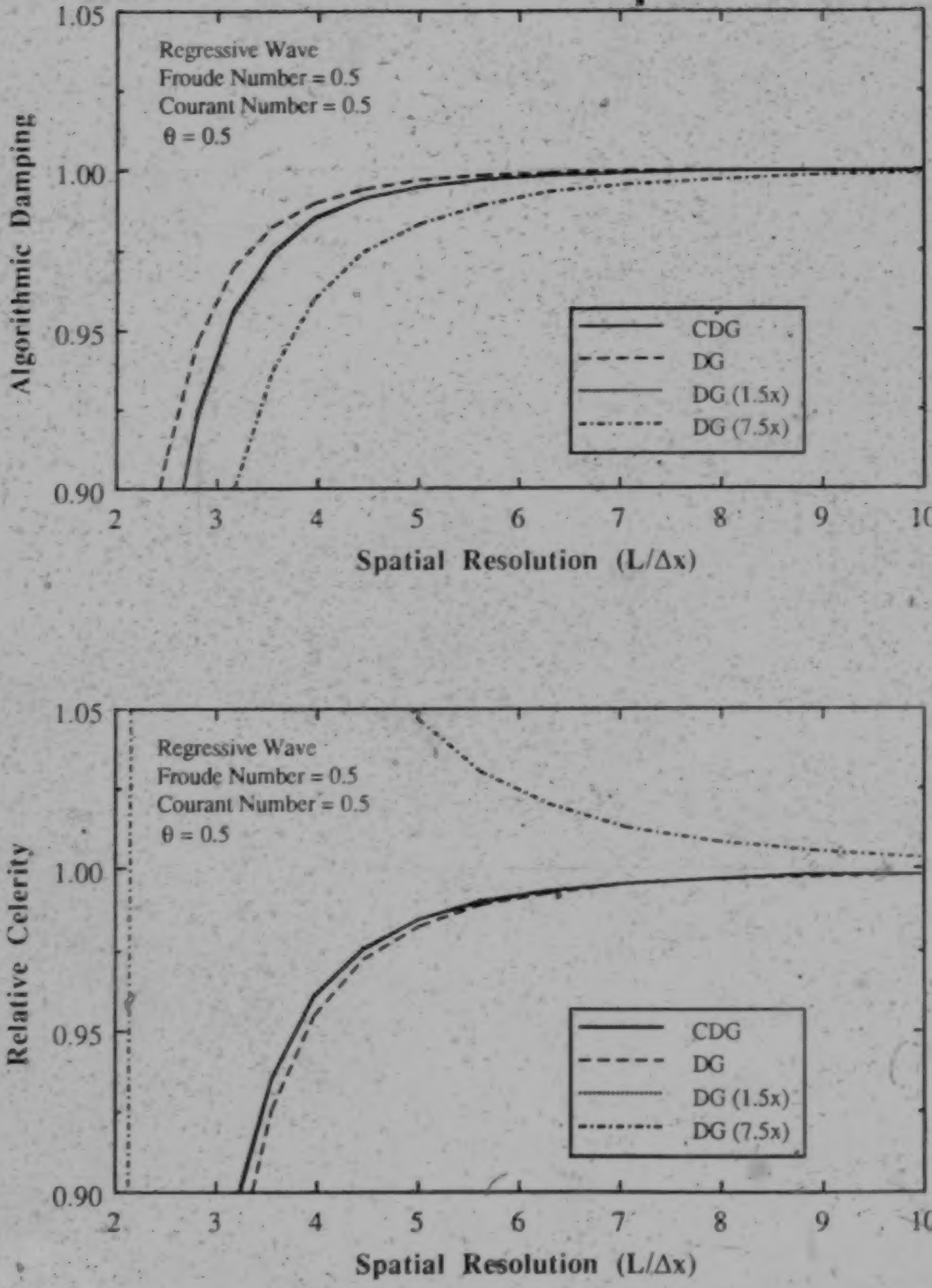
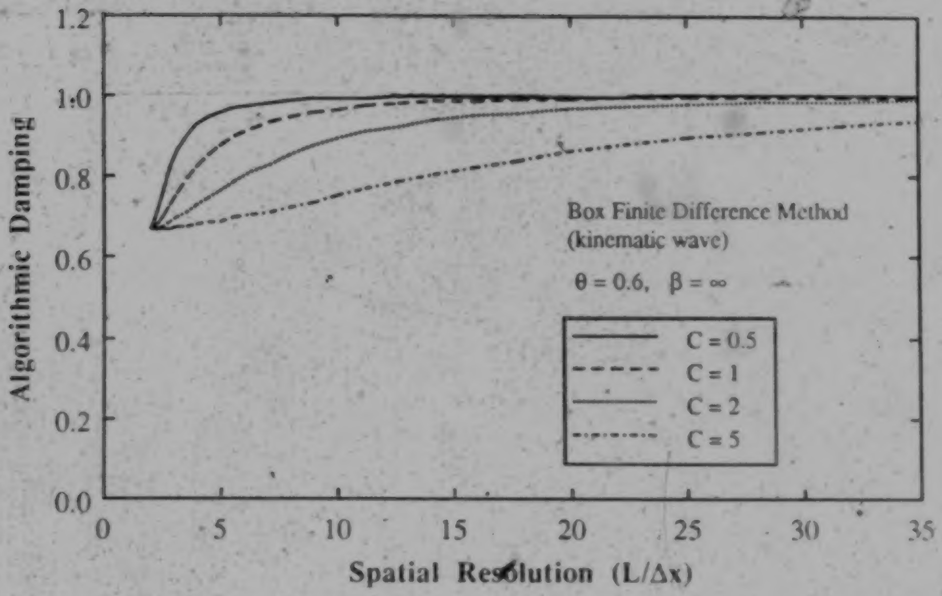


Figure 4.44 Effect of increasing the upwinding coefficient, ω, for the Dissipative-Galerkin Method, on the results of stability analysis, C = 0.5, **F** = 0.5, θ = 0.5 (regressive wave).



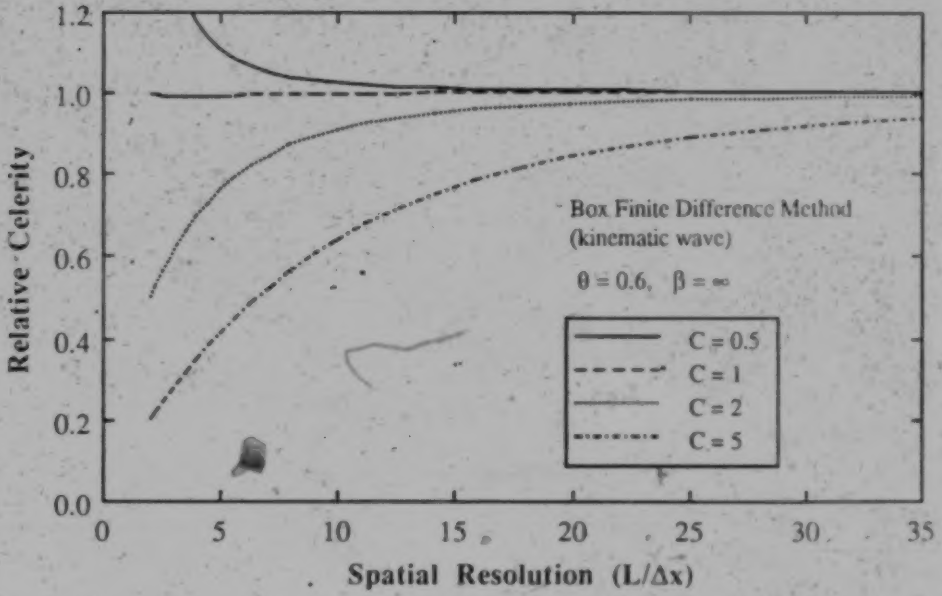


Figure 4.45 Results of stability analysis, Box Finite Difference Scheme, $\beta = \infty$, $\theta = 0.6$ (kinematic wave).

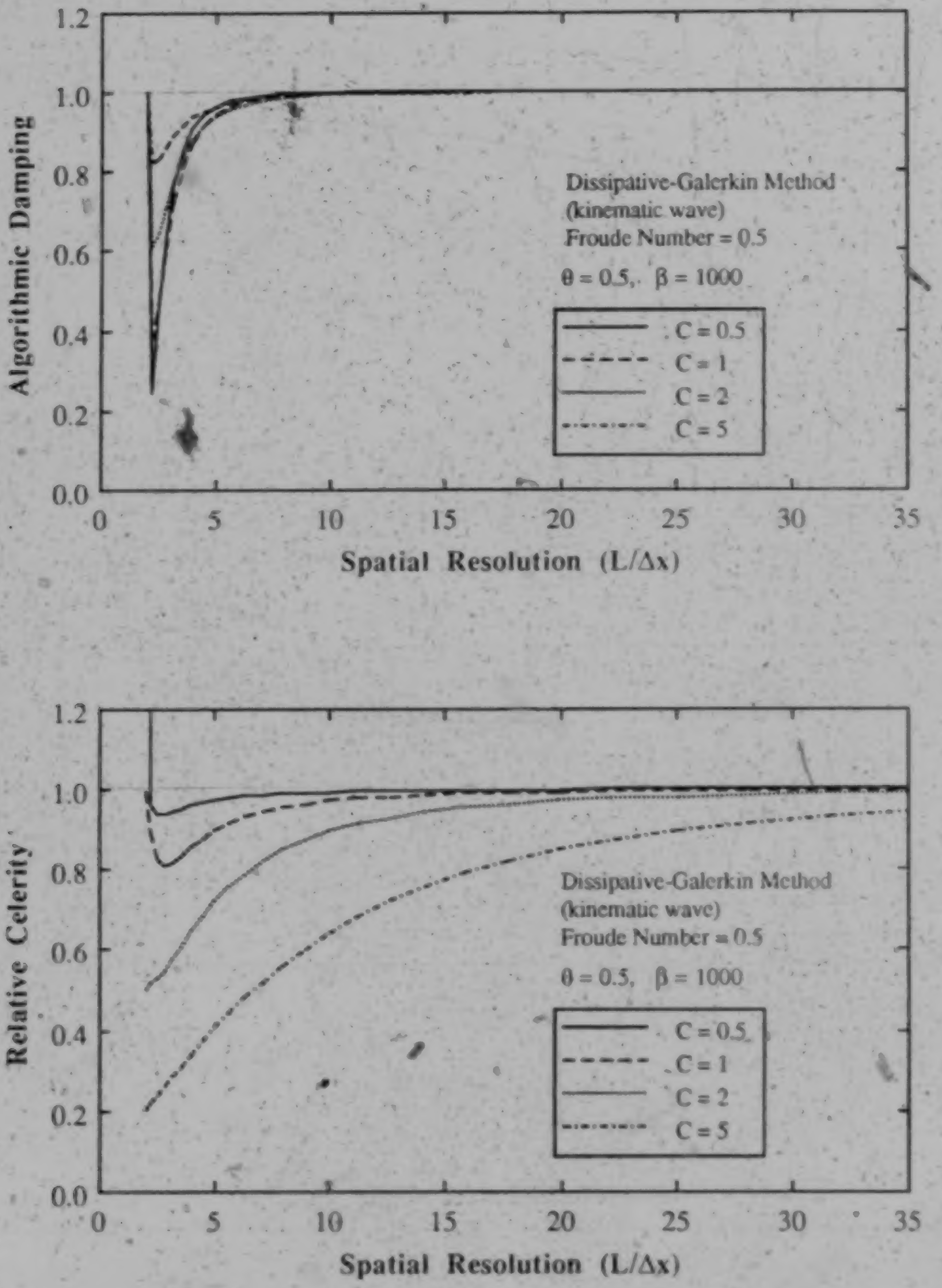


Figure 4.46 Results of stability analysis, Dissipative-Galerkin Scheme, β =.1000,
F = 0.5, θ = 0.5 (kinematic wave)

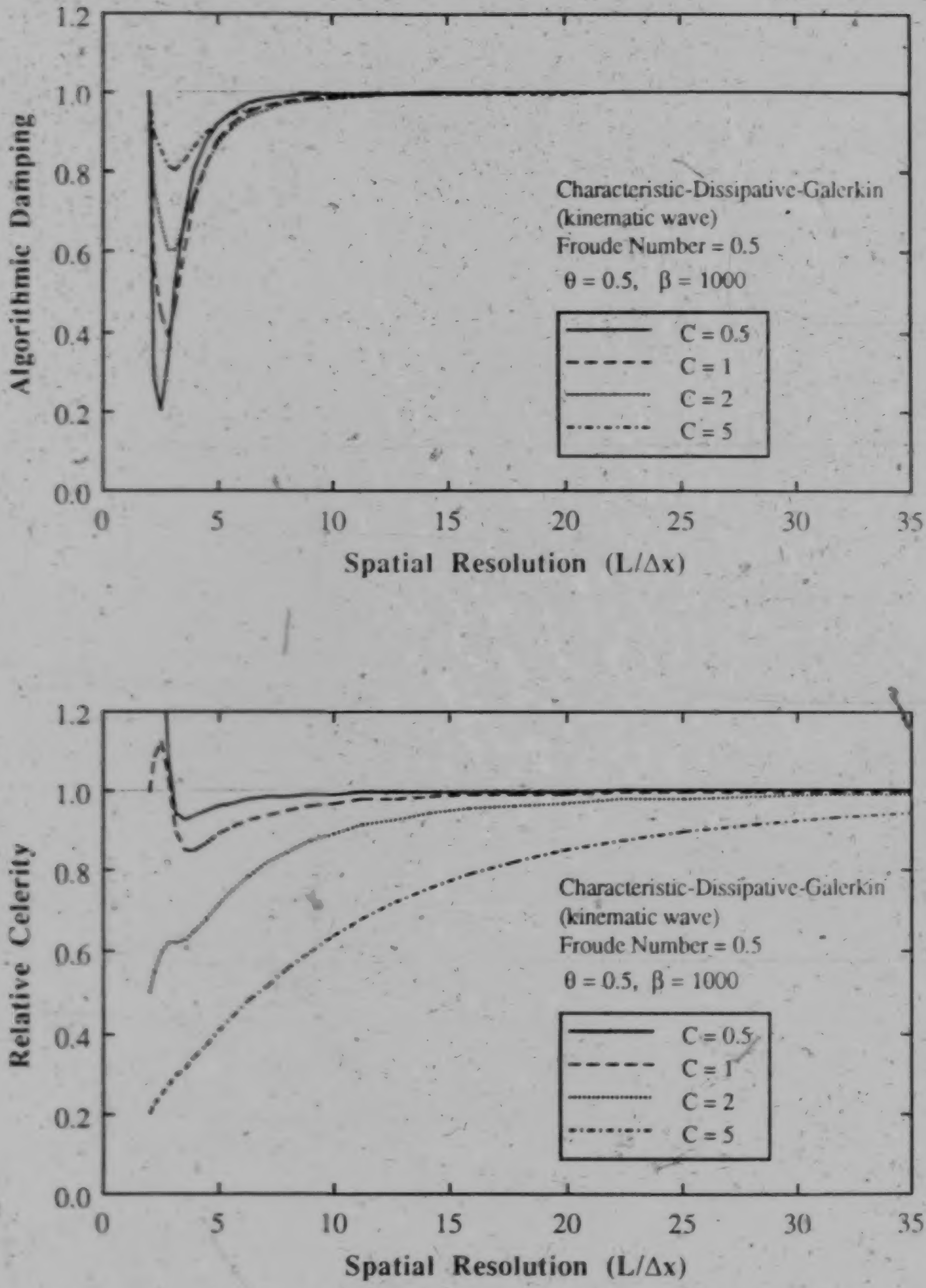


Figure 4.47 Results of stability analysis, Characteristic-Dissipative-Galerkin Scheme, β = 1000, F = 0.5, θ = 0.5 (kinematic wave)

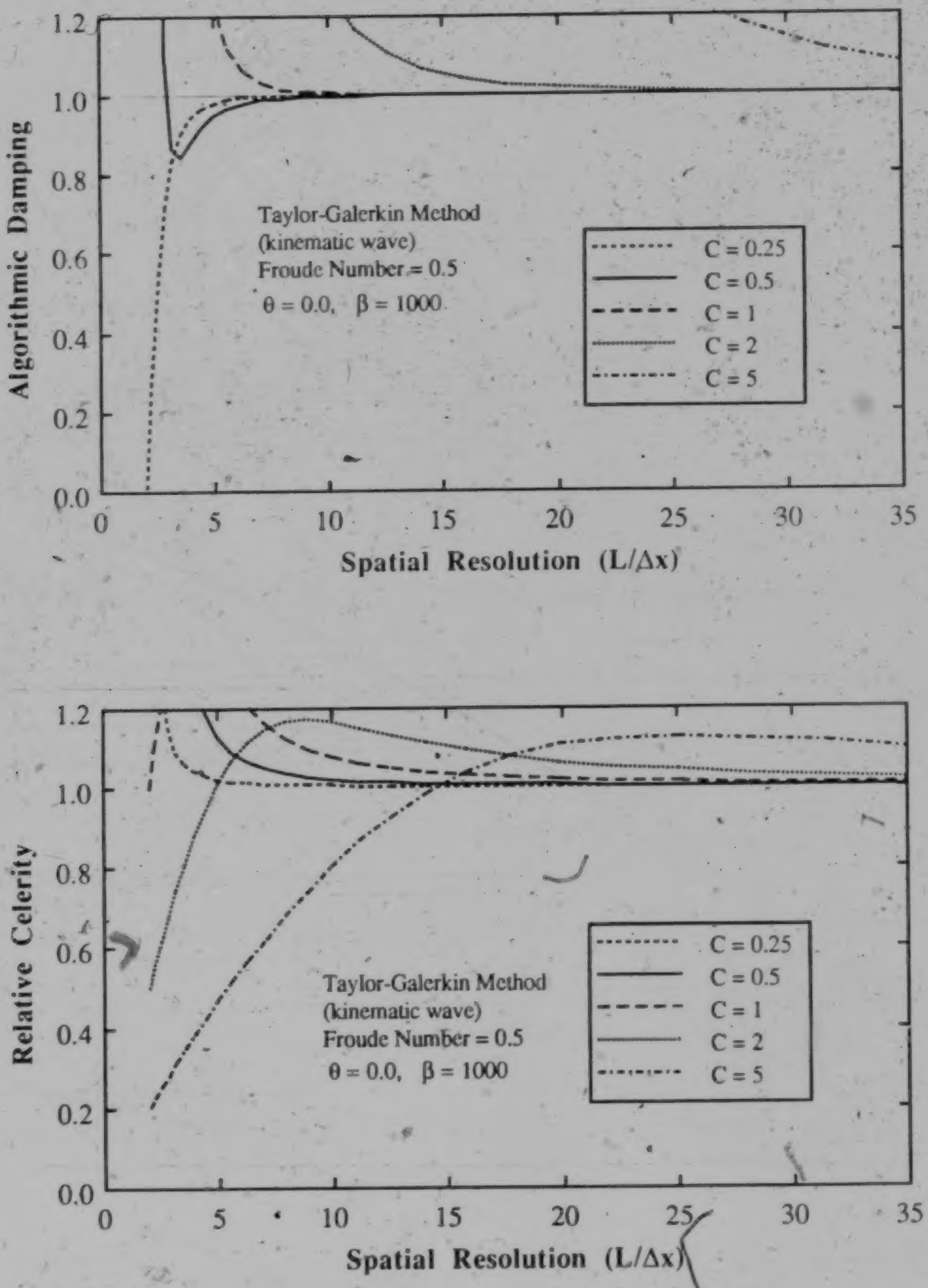


Figure 4.48 Results of stability analysis, Taylor-Galerkin Scheme, β = 1000, θ = 0.0 (kinematic wave)

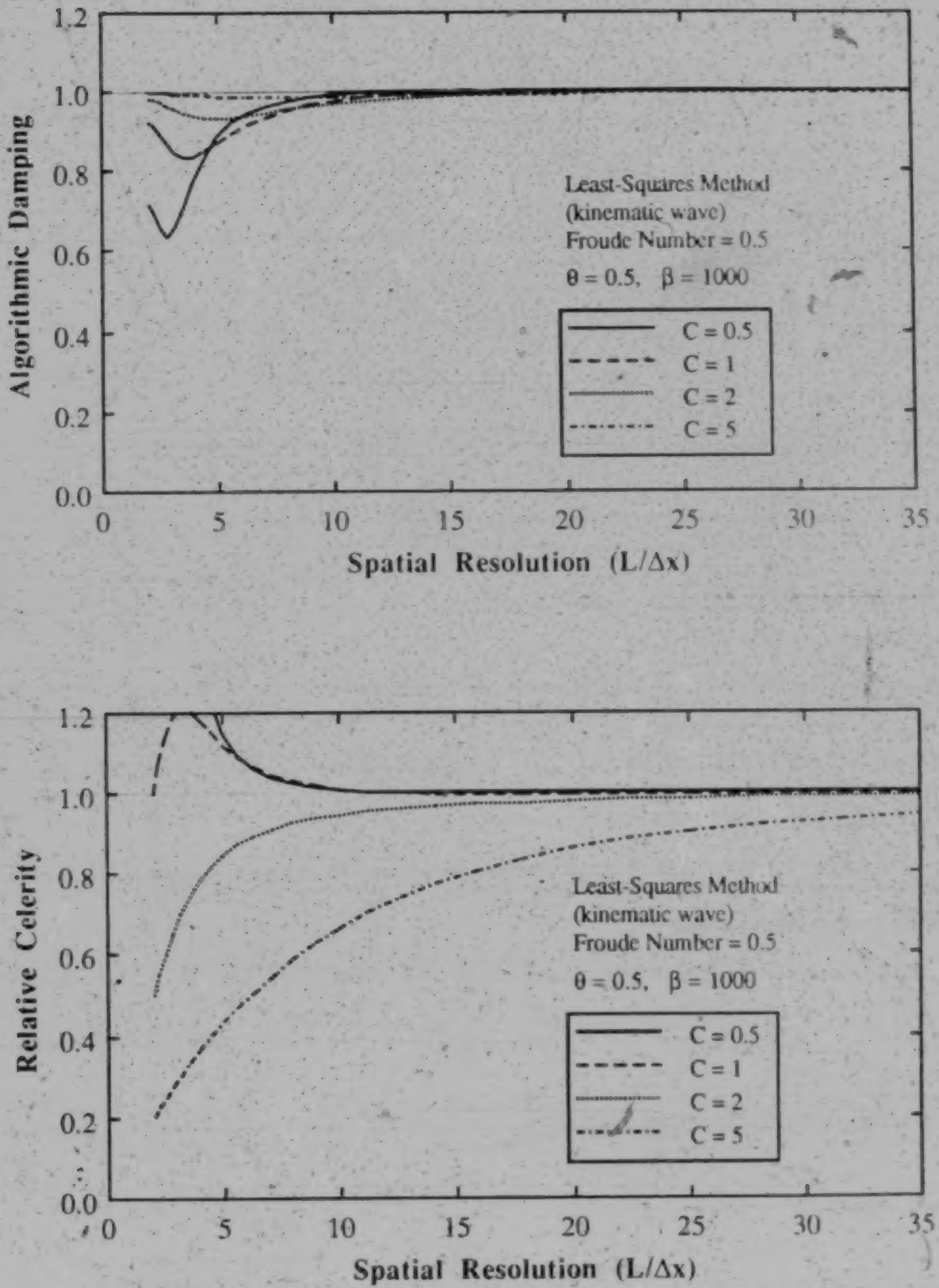


Figure 4.49 Results of stability analysis. Least-Squares Scheme, $\beta = 1000$, $F = 0.5$, $\theta = 0.5$ (kinematic wave)

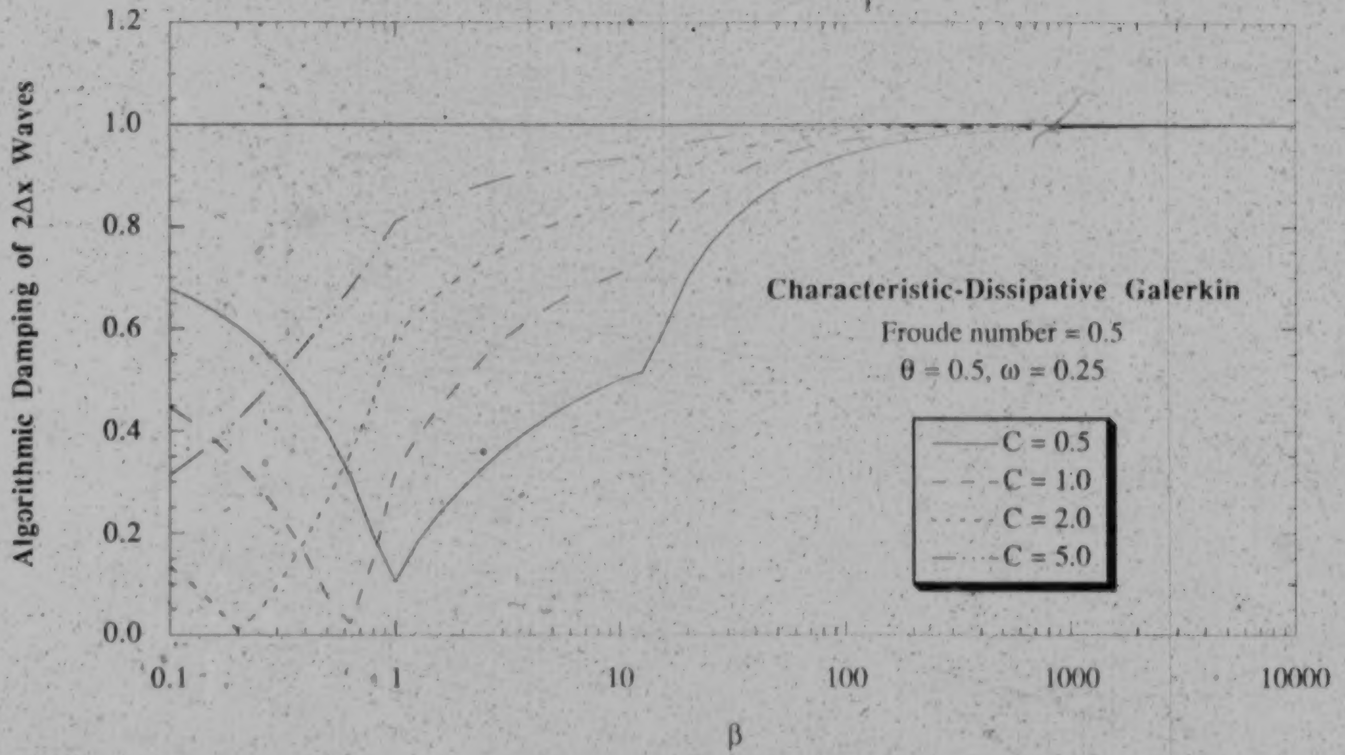


Figure 4.50 Algorithmic damping of $2\Delta x$ waves for the CDG scheme as a function of β, $\theta = 0.5$, $F = 0.5$, $\omega = 0.25$.

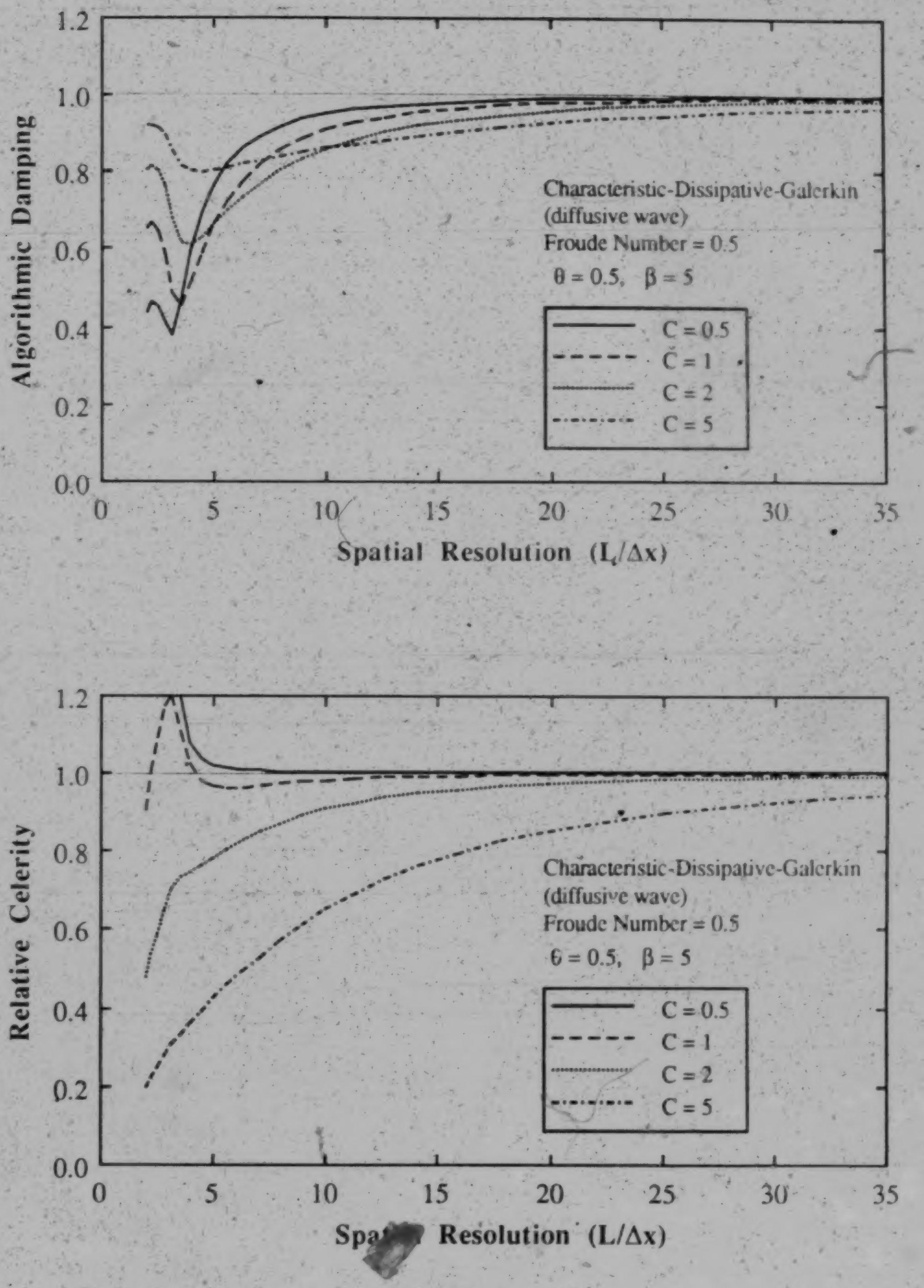


Figure 4.51 Results of stability analysis, Characteristic-Dissipative-Galerkin Scheme, $\beta = 5$, F = 0.5, $\theta = 0.5$ ([illegible] wave)

5. NUMERICAL ANALYSIS

5.1 INTRODUCTION

In this chapter an objective and critical analysis of the various schemes is conducted through the use of exhaustive and rigorous tests. This includes the examination of both steady and unsteady flows including the extremes of fully dynamic and kinematic waves.

Except as noted, all tests were conducted using the conservation form of the St. Venant equations with area and discharge as the dependent variables. However, the non-conservation form of the equations were used in the evaluation of the upwinding matrix in the implementation of the Petrov-Galerkin schemes. Details of the implementation of the finite element schemes are outlined in Appendix B.

The investigation of the CDG scheme was extended to examine a symmetric form of the St. Venant equations to determine whether significant cross-upwinding had occurred. As this necessitated the use of a non-conservation system, a comparison of conservative and non-conservative non-symmetric formulations was also done for this method.

For the DG and CDG schemes, the upwinding parameter, ω, was taken to be 0.50 for the steady flow tests, while a value of 0.25 was selected for unsteady flow problems. However, these values were varied in certain cases to examine the sensitivity of the results to this parameter. As discussed in Chapter 3, the upwinding for the TG scheme and the LS method both depend on the time step increment (and the level of implicitness in the case of the LS method), therefore the amount of numerical diffusion added for these schemes was problem dependent.

In terms of the iterative scheme, runs were conducted using a number of variations. This included using the [illegible] matrix to approximate the Jacobian, as discussed in Chapter [illegible] as well as successive approximations including a complete analytical Bubnov-Galerkin Jacobian as well as approximations to the full analytical DG and CDG Jacobians. These are detailed in Appendix B, as well. Unless otherwise noted, the runs for the results presented were conducted using the BG Jacobian which was determined to be both sufficient yet beneficial enough to warrant the extra computational effort, in most cases. With a specified tolerance of 10^{-5} (as defined in equation [3.30]) the implicit finite element schemes typically converged in 3 to 5 iterations. Another variation in the iteration process was the question of whether or not to update the upwinding matrix every iteration or every time step. In many cases this did not affect the convergence rate or the ultimate solution. However, in some cases one or the other approach was essential in achieving convergence. In general, it may be assumed that the results shown were based on updating every time step, unless specified to the contrary.

Details of the initial and boundary conditions are provided with each test with one notable exception. The TG scheme, implemented as an equivalent Bubnov-Galerkin formulation of the system given by equation [3.74], required additional boundary conditions. For the purpose of this investigation, this was achieved by specifying both area and discharge at each boundary for all tests.

5.2 STEADY FLOW TESTS

5.2.1 Introduction

The first set of tests used in this analysis were designed to examine the convergence ability of the various schemes. Ideally, this would be facilitated by a quick progression to the ultimate steady state solution through a coarse tolerance and large time step increments. In addition, it was desirable to examine the consistency of the method is converging to the same solution given a differing time step increment or level of implicitness. Both dynamic and friction dominated problems were of interest. The first test involves a steady hydraulic jump, while the second examined the behavior of the various schemes in the evaluation of gradually varied flow profiles.

5.2.2 Hydraulic Jump

5.2.2.1 General

This test, adapted from Katopodes (1984), involved the transition from a supercritical to a subcritical flow in a horizontal, frictionless channel. Thirty nodes, spaced at 5 m intervals, were used in the analysis. The discharge was initially set to 5 cubic metres per second (cms) at all nodes. Through the upstream half of the domain the initial supercritical flow depth was set to 1.008 metres (m) while a subcritical flow of depth of 1.80 m was set for the downstream half. The boundary conditions provided were area and discharge at the upstream end, and area at the downstream end.

This test could not be run for the BFD scheme due to the combination of subcritical and supercritical flow. For the implicit schemes: BG, DG, CDG and LS, the test was first run semi-implicit at a Δt of 0.0072s, corresponding to a Courant number of 0.01. It was then run fully implicit at a Δt of 0.072s, corresponding to a Courant number of 0.1. The explicit Taylor-Galerkin scheme was run at a Δt of 0.0036s, corresponding to a Courant number of 0.005. A strict tolerance of 10^{-5} was used in the iterative process.

A comparison of the maximum negative deviations is presented in Table 5.1 to aid in the discussion.

5.2.2.2 Discussion of Results

Figure 5.1 shows the results for the BG method after 20 time steps (t = 0.144s) using $\theta = 0.5$ and $C = 0.01$. For comparison purposes, the initial conditions are shown as well.

Table 5.1 Maximum negative deviations for the hydraulic jump test.

a	C	t (s)	Value	BG	TG†	TG (lumped)†	DG††	DG†††	CDG††	CDG††††	LS
[illegible]	0.01	0.144	ΔH	-0.00139	-0.00137	-0.00032	-0.00001	-0.00014	-0.00897	-0.00412	-0.00336
			ΔQ	-0.09206	-0.09195	-0.04264	-0.07693	-0.00149	-0.07790	-0.08570	-0.09187
[illegible]	0.01	0.72	ΔH	-0.02946	-0.02939	-0.00879	-0.02639	-0.00017	-0.01286	-0.01610	-0.05200
			ΔQ	-0.39615	-0.38801	-0.15290	-0.14926	-0.00314	-0.14423	-0.17925	-0.37662
[illegible]	0.1	0.72	ΔH	-0.02955	-0.02939	-0.00879	-0.02800	-0.00018	-0.01159	-0.01538	-0.14054
			ΔQ	-0.38861	-0.38801	-0.15290	-0.14136	-0.00302	-0.13401	-0.16447	-0.19303
[illegible]	0.1	7.2	ΔH	-0.60461	-0.59412	-0.27478	-0.19499	-0.00009	-0.00768	-0.07117	-0.32058
			ΔQ	-2.85560	-2.41387	-0.84581	-0.26261	-0.00166	-0.05348	-0.14942	-0.63913

† explicit runs, $\Delta t = 0.0036s$, $C = 0.005$

†† upwinding coefficient = 0.5

††† upwinding coefficient = 0.5 | U + c |

†††† upwinding coefficient = 0.25

Although the shock still appeared well preserved at this time, 2Δx oscillations in the discharge were observed. A test was also conducted for a non-conservation form of the equations for this study and results were less than 1 percent different from those shown in Figure 5.1. The results after 100 time steps ($t = 0.72s$) can be seen in Figure 5.2 which shows not only increased oscillations in the discharge, but in the depth as well. Figure 5.3 illustrates the results at the same time, based on a $\theta = 1.0$ and $C = 0.1$. As Table 5.1 indicates; results were not significantly different for the two runs at $t = 0.72$ s. Finally Figure 5.4 shows the results for the implicit run after 100 steps ($t = 7.2s$) where significant deterioration of the solution had occurred, illustrating the inability of the BG scheme to damp 2Δx disturbances as was predicted by the linear stability analysis.

Figure 5.5 shows the results for the TG method after 40 time steps ($t = 0.144s$). At this time, the shock still appeared well preserved. However, like the BG scheme, discharge oscillations in the vicinity of the shock were observed. In fact, as Table 5.1 illustrates, the results were nearly identical to the BG method. Similar results are seen in Figure 5.6 after 200 time steps ($t = 0.72s$). Figure 5.7 shows the results after 2000 steps ($t = 7.2s$) where significant deterioration of the solution had occurred. Figures 5.8 through 5.10 show the results obtained by lumping the mass matrix for the TG scheme. Although this did not lead to a stable solution, the results were noticeably better than the consistent TG formulation, as shown in Table 5.1.

The initial runs for the DG method were based on an upwinding coefficient , ω, of 0.50. As seen in Figure 5.11, after 20 time steps ($t = 0.144s$) using $\theta = 0.5$ and $C = 0.01$, the shock still appeared well preserved. Also, as Table 5.1 shows, the discharge oscillations were smaller than those observed for the BG and TG schemes. In Figure 5.12, after 100 time steps ($t = 0.72s$) small oscillations can be seen in both the depth and discharge. Furthermore, as Figure 5.13 illustrates, the results at $t = 0.72s$

C = 0.01. Finally, Figure 5.14 shows the results for the implicit run after 100 steps (t = 7.2s) where, although the shock was preserved, increased oscillations were apparent. This is confirmed by Table 5.1. However, as expected from the linear stability analysis, this scheme did have the ability to damp 2Δx waves, therefore the solution was noticeably more stable than the BG results. To examine the DG method as it was run by Katopodes (1984), the value of the upwinding coefficient was increased by a factor of |U+c|. As seen in Figures 5.15 through 5.18, this resulted in a perfectly stable and accurate preservation of the jump, with negligible oscillations in depth and discharge. Furthermore, as Table 5.1 shows, the maximum negative deviations actually decreased in time when the upwinding coefficient was increased.

The CDG scheme was initially run using an upwinding coefficient of 0.50, as well. Figure 5.19 shows the results after 20 time steps (t = 0.144s) using θ = 0.5 and C = 0.01. The shock was well preserved at this time although small oscillations in the discharge were observed. As shown in Table 5.1 the maximum negative deviation was slightly larger than that obtained in the corresponding DG run. The results after 100 time steps (t = 0.72s) can be seen in Figure 5.20 which shows only minor changes in the jump profile, while oscillations in the discharge had increased. Figure 5.21 illustrates nearly identical results at t = 0.72s, based on a θ = 1.0 and C = 0.1, as Table 5.1 illustrates. The maximum negative discharge deviation in this case was smaller than that observed in the corresponding DG run. Finally Figure 5.22 shows the results for the implicit run after 100 steps (t = 7.2s) where oscillations in the discharge had in fact decreased in the vicinity of the jump, which was itself still quite well preserved. There was, however, some slight decrease in discharge noted in the downstream, subcritical, reach.

Because the ultimate intent was to develop a model suitable for both steady and [illegible] flows, it was desirable to examine the behavior of the CDG scheme coefficient of 0.25, are shown in Figures 5.23 through 5.26. Although the results indicated that the magnitude of the discharge deviation in the vicinity of the shock would be increased, as seen in Table 5.1, these effects were localized and did not destabilize the solution. In addition, the jump was still well preserved indicating that the CDG scheme could be run at using the same upwinding coefficient for both steady and unsteady flow, if necessary.

Results for the Least-Squares method are presented in Figures 5.27 to 5.30. After 20 time steps (t = 0.144s) using $\theta = 0.5$ and $C = 0.01$, the results were quite similar to the corresponding BG run as seen from Table 5.1. Results after 100 time steps (t = 0.72s) were also similar to those for the BG, as seen in Figure 5.28. Results at t = 0.72s based on a $\theta = 1.0$ and $C = 0.1$ were not the same as for $\theta = 0.5$ and $C = 0.01$, given that both θ and Δt had increased. However, it is interesting to note that this increased upwinding reduced the magnitude of the discharge oscillations at the expense of the depth profile, as Table 5.1 illustrates. This is further seen in Figure 5.30, for the implicit run after 100 steps (t = 7.2s) where significant deterioration of the solution had occurred, though not only in the form of $2\Delta x$ oscillations as in the case of most other methods.

5.2.2.3 Spectral Analysis

It is interesting to note that several methods, indicated by the linear stability analysis to be capable of damping $2\Delta x$ disturbances, still encountered instabilities in the vicinity of this shock. Evidently, these instabilities, associated with the non-linear transfer of energy between wavelengths were created quicker than they were damped. To further examine the nature of the problems encountered, a spectral analysis was employed. This was achieved by taking a Fourier Transform of the flow depth profile at both t = 0s and t = 7.2s for each method ($\theta = 1.0$, $C = 0.1$, except in the case of the TG run). The results of this analysis for the BG and TG schemes are shown in Figure 5.31, in which E would be interpreted as some measure of energy density. As might be expected they are quite similar, illustrating an accumulation of the short wavelengths (particularly $2\Delta x$ waves) over time. It appears that waves aliased to these shorter wavelengths could not be dissipated at the same rate at which they accumulated. In contrast, Figure 5.32 shows the results for the DG and CDG schemes for $\omega = 0.50$. Their selective damping of high frequency wavelengths is apparent, particularly for the CDG method, where no aliasing effects are observed. Finally, Figure 5.33 illustrates the results for the Least-Squares method in which aliasing is seen though not at the $2\Delta x$ wavelength. This corresponds with the profile illustrated in Figure 5.30. Nevertheless, like the DG and CDG schemes, damping is apparent.

5.2.3 Transition from a Mild to a Steep Channel

5.2.3.1 General

This test involved the calculation of gradually varied flow through the transition from a mild to a steep channel. The channel used was 10 m wide, with an upstream slope of 0.0025 and a downstream slope of 0.04. The effective bed roughness, k, was set to 0.2 m throughout the domain which consisted of 65 elements, each 10 m long, for a total of 66 nodes. The uniform flow depths for the mild and steep channel were 3.64 m and 1.44 m, respectively. The discharge was initially set to 100 cms while the initial depth was set to 3.5 m, as seen in Figure 5.34 (a). Because of the subcritical flow in the upstream channel only one upstream boundary condition was specified; while at the downstream end, no boundary condition was required. The exception was the TG scheme, which as discussed above required area and discharge to be specified at both boundaries. Based on the results of the other tests, the downstream area was set to the supercritical flow depth for the TG scheme, as seen in Figure 5.34 (b). Again the BFD scheme could not be included in these tests due to the combination of subcritical and supercritical flow.

5.2.3.2 Discussion of Results

The tests were run using time step increments of 0.50s, 0.25s and 0.125s. These corresponded to Courant numbers of about 0.35, 0.17 and 0.09, respectively, based on the supercritical uniform flow velocity. Larger time step increments were attempted, however, stable solutions could not be achieved. A strict tolerance of 10^{-5} was required for all tests to avoid numerical instabilities.

Figure 5.35 shows the results for the BG scheme after 1000 time steps using a time step increment of 0.25 s. The solution was particularly sensitive to time step and tolerance and an increase in either quickly lead to instabilities. As Figure 5.35 shows, a relatively smooth profile was achieved, with the discharge deviation increasing gradually in the downstream direction to a maximum of 1 percent.

Figure 3.36 shows the results for the TG scheme after 310 time steps, using a time step increment of 0.25. As the figure shows, this scheme provided a stable flow profile, in which the results were within 0.1 to 0.15 m of the BG profile. However, a significant deviation in the discharge was noted, particularly at the downstream boundary where it reached a maximum of 9.8 percent. It should be noted again that the discharge was fixed at the downstream boundary for this scheme, as extra boundary conditions were required. As seen in Figure 3.37, results were similar for the lumped TG although the maximum discharge deviation was just over 10 percent in this case.

For the DG scheme, the maximum stable time step was 0.25 s. Figure 5.38 illustrates the solution obtained after 2000 time steps using an upwinding coefficient of 0.50. The results obtained were quite smooth with only a slight oscillation observed in the discharge at the downstream boundary, corresponding to an error of less than 1 percent. The depth profile was within 0.01 m of the corresponding BG results.

Results for the CDG scheme were quite similar to those for the DG method, as Figure 5.39 shows. In this case, although the run was stable at a time step of 0.50 s, 2000 time steps were still required. In this case there was no significant discharge oscillation at the slope break (less than 0.05 percent), while the depth profile was within 0.005 m of the results obtained for the DG scheme.

The Least-Squares method could not achieve a stable solution for this test. Discharge oscillations growing from the downstream boundary destroyed the solution quickly in every case. An example is shown in Figure 5.40, which illustrates the solution for $\Delta t = 0.125$s, just before the solution failed.

5.2.4 Comparison of Methods

The ability of most of these finite element schemes to converge to a stable and consistent solution, illustrates their applicability to steady open channel flow problems. Of the methods considered, the Petrov-Galerkin DG and CDG were the most promising, particularly in terms of the discharge accuracy. The sensitivity of all the methods to time step and tolerance is significant. Larger values of the tolerance lead to instabilities which quickly destroyed the solution, while larger time steps either destabilized the solution or caused a convergence to the subcritical root and a physically unrealistic solution.

5.3 UNSTEADY FLOW TESTS

5.3.1 Introduction

These tests were designed to examine the ability of the various schemes to model unsteady flow with both stability and accuracy. Where possible, the exact solution was provided to facilitate the evaluation and comparison of the various schemes. In addition, it was desirable to examine the consistency of the various methods by varying the time step increment or the level of implicitness. Both dynamic and friction dominated problems were of interest. A strict tolerance of 10^{-5} was used in the iterative process for all unsteady flow tests.

5.3.2 Disturbance Propagation in a Horizontal Frictionless Flow

5.3.2.1 General

The purpose of this set of tests was to examine the ability of the various schemes to propagate disturbances accurately. By eliminating the friction and slope terms from consideration, exact solutions based on the method of characteristics could be obtained easily, thus facilitating this evaluation. Initially, a test was run using the linear equations so as to provide for a direct comparison with the linear stability analysis as well as to point out any effect of the non-linearities in the subsequent test. For the non-linear case, a variety of Froude numbers were examined in order to evaluate the various schemes over as broad a spectrum as possible.

In all cases, the geometry consisted of a unit width section of a horizontal, frictionless channel. The 99 elements were each 5 m long for a total of 100 nodes. The boundary conditions for the subcritical flow runs were given as discharge at the upstream and downstream ends, while for the critical and supercritical flows, both discharge and area were specified at the upstream end. The initial depth condition was taken as 3.5 m , except in the vicinity of the disturbance(s). Discharge was specified based upon the Froude number selected for the test.

5.3.2.2 Linear Perturbation - Froude Number = 0.0

Figure 5.41 illustrates the initial conditions specified for this test. In addition the discharge was set to zero throughout the domain. The implicit schemes were run at time step increments of 0.8533 s and 0.42665 s, corresponding to Courant numbers of 1.0 and 0.5, respectively. The TG scheme was run at $C = 0.5$ and $C = 0.25$ ($\Delta t = 0.213325$ s).

With the exception of the BFD scheme which was run at $\theta = 0.6$, all implicit schemes were run for $\theta = 0.5$. Results were presented at t = 25.6 s, after the disturbance had split into two and each had travelled exactly 150 m. The peaks, each of depth 4.62211 m were located at 95 and 395 m respectively. The results for the numerical tests are summarized in Table 5.2, while Table 5.3 provides the maximum negative depth deviations.

Figure 5.42 shows the results obtained using the BFD scheme. As expected from the linear stability analysis (Figure 4.4), the results for C = 1.0 illustrate that the peak phase speed was exact, though the solution was quite dissipative. As Table 5.2 shows, both peaks were 0.30 m low, a difference of 6.5 percent. Although at C = 0.5 the peak was less damped (4.6 percent low), shorter wavelengths were propagated too fast resulting in leading disturbances, as illustrated in Table 5.3.

The results for the BG scheme can be seen in Figure 5.43. The linear stability analysis predicted that no algorithmic damping would occur, though the shorter wavelengths would be propagated too slowly (Figure 4.7). This can be seen in Figure 5.43 (a), for C = 1.0 where the trailing disturbances were accumulating behind both waves and, to a lesser extent in part (b) for C = 0.5. This lead to a decrease in the wave peak of 1.2 and 1.5 percent for C = 1.0 and 0.5, respectively, as Table 5.2 shows.

As stated above, the TG scheme was run at Courant numbers of 0.5 and 0.25. The linear stability analysis predicted that at these values, there would be very selective damping of the high frequency waves (Figure 4.13). However, a phase speed error would be expected for these short wavelengths, particularly at C = 0.5. This can be seen in Figure 5.44 and Table 5.2, where leading disturbances caused a noticeable decrease in the magnitude of the peaks at C = 0.5 and both leading and trailing disturbances were observed at C = 0.25. As Table 5.3 shows, this lead to a maximum negative depth deviation

Table 5.2 Wave peaks for the linear propagation test (semi-implicit).

Regressive Waves

C	Value	BFD*	BG	TG†	TG (lumped)†	DG††	CDG††	LS
1	H	4.32038	4.56618			4.55643	4.56074	4.28002
	x	95	100			100	100	95
0.5	H	4.41112	4.55509	4.37723	4.37674	4.56995	4.57275	4.35564
	x	95	100	95	95	95	95	95
0.25	H			4.45647	4.44337			
	x			95	100			

Progressive Waves

C	Value	BFD*	BG	TG†	TG (lumped)†	DG††	CDG††	LS
1	H	4.32038	4.56618			4.56071	4.56074	4.28002
	x	395	390			390	390	395
0.5	H	4.41113	4.55509	4.37723	4.37674	4.57273	4.57275	4.35564
	x	395	390	395	395	395	395	395
0.25	H			4.45647	4.44337			
	x			395	390			

* Theta = 0.6 for BFD

† explicit runs

†† upwinding coefficient = 0.25

Table 5.3 Maximum negative depth deviations for the linear propagation test (semi-implicit).

C	BFD*	BG	TG†	TG (lumped)†	DG††	CDG††	LS
1	0.00000	-0.09312			-0.08624	-0.08525	-0.05601
0.5	-0.02204	-0.07028	-0.09789	-0.09487	-0.04205	-0.04074	-0.04750
0.25			-0.02829	-0.14270			

* Theta = 0.6 for BFD

† explicit runs

†† upwinding coefficient = 0.25

comparable in magnitude to the BG results. Results for the lumped TG are shown in Figure 5.45, where the phase error appeared to consist of trailing rather than leading waves. In this case, unlike the steady flow results, the lumped TG results were worse than those for the consistent TG, particularly at C = 0.25.

The results for the DG scheme are shown in Figure 5.46 for an ω = 0.25 and as expected from the linear stability analysis (Figure 4.16), very little damping of the peaks occurred (1.1 to 1.4 percent). However, some trailing disturbances were noted, particularly at a Courant number of 1.0, as noted in Table 5.3. Similar results obtained for the CDG scheme, as seen in Figure 5.47, also agreed with the predictions of the linear stability analysis (Figure 4.22).

Figure 5.48 illustrates the results for the LS method where the less selective damping of this scheme had significantly deteriorated the peak, especially at C = 1.0 where it was 7.4 percent below the exact solution. In addition, trailing waves due to phase error were observed, as predicted by the stability analysis (Figure 4.30).

Based on these results, it appeared that the DG and CDG schemes were clearly superior to the other methods in achieving stable and accurate results. Furthermore, it was found that all the finite element methods, save the LS scheme, exhibited better peak preservation than the BFD method. It is interesting to note that the results for the DG scheme were different for the two waves, as seen in Table 5.2. However, for all of the other methods results for the progressive wave were identical to those for the regressive wave.

5.3.2.3 Non-Linear Perturbation - Froude Number = 0.0

To facilitate the computation of exact solutions for the non-linear problems, the initial conditions were set as two identical disturbances, one progressive and one regressive.

This approximated a disturbance similar to that used in the linear test, as if some small amount of time had passed. Figure 5.49 illustrates the initial conditions specified for this particular test. The implicit schemes were run at time step increments of 0.5780 s and 0.2890 s, corresponding to Courant numbers of 1.0 and 0.5, respectively. The TG scheme was run at C = 0.5 and C = 0.25 ($\Delta t = 0.1445$ s). With the exception of the BFD scheme which was run at $\theta = 0.6$, all implicit schemes were run for $\theta = 0.5$. Results were presented at t = 17.3 s, after the two disturbances had travelled exactly 150 m. The peaks, each of depth 4.70 m were located at 65 and 430 m, respectively. The results for the numerical tests are summarized in Table 5.4, while Table 5.5 provides the maximum negative depth deviations.

Figure 5.50 shows the results obtained for the BFD scheme. The tendencies observed were similar to those of the linear test results, although the peak was better preserved, particularly in the case where C = 0.5. However, as Table 5.4 shows, these results are still 4 to 5 percent low). For the BG method, Figure 5.51 illustrates a similar trend with somewhat improved results compared to the linear problem. The results for the TG and lumped TG are provided in Figure 5.52 and 5.53, respectively. Again the superiority of the consistent TG over the lumped version was evident. Also, as for the BFD and BG much less deterioration of the peak due to phase error was noted (0.5 to 1.4 percent), compared to the linear test.

As in the previous test, both the DG and CDG performed extremely well, particularly at C = 0.5, as seen in Figures 5.54 and 5.55, respectively. To examine the effect of the increased upwinding coefficient used in the steady flow analysis, the DG was run using a value of $\omega = 0.50\,|U+c|$. The results, shown in Figure 5.56(a), show a 2.2 percent decrease in the peak, associated with a leading phase error, compared to 1.5 percent for $\omega = 0.25$ at C = 1.0. Similarly, the CDG scheme was run with the value of ω increased

Table 5.4 Wave peaks for the non-linear propagation test for Froude number = 0.0 (semi-implicit).

Regressive Waves

C	Value	BFD*	BG	TG†	TG (lumped)†	DG††	DG†††	CDG††	CDG††††	LS
1	H	4.45973	4.64754			4.63089	4.59428	4.63729	4.62729	4.47952
	x	75	70			70	70	70	70	75
0.5	H	4.49863	4.64209	4.60755	4.63286	4.63725		4.64509		4.52304
	x	70	65	65	75	70		70		75
0.25	H			4.65451	4.67629					
	x			65	75					

Progressive Waves

C	Value	BFD*	BG	TG†	TG (lumped)†	DG††	DG†††	CDG††	CDG††††	LS
1	H	4.45973	4.64754			4.63729	4.59428	4.63729	4.62729	4.47952
	x	420	425			425	425	425	425	420
0.5	H	4.49862	4.64209	4.60755	4.65286	4.64507		4.64509		4.52304
	x	425	430	430	420	425		425		420
0.25	H			4.65451	4.67629					
	x			430	420					

* Theta = 0.6 for BFD

† explicit runs

†† upwinding coefficient = 0.25

††† upwinding coefficient = 0.5 | U + c |

†††† upwinding coefficient = 0.50

Table 5.5 Maximum negative depth deviations for the non-linear propagation test for Froude number = 0.0 (semi-implicit).

C	BFD*	BG	TG†	TG (lumped)†	DG††	DG†††	CDG††	CDG††††	LS
1	-0.01234	-0.04225			-0.03294	-0.02876	-0.03260	-0.02807	-0.05697
0.5	-0.11873	-0.02367	-0.11245	-0.05107	-0.01674		-0.01637		-0.06414
0.25			-0.03422	-0.05226					

* Theta = 0.6 for BFD

† explicit runs

†† upwinding coefficient = 0.25

††† upwinding coefficient = 0.5 | U + c |

†††† upwinding coefficient = 0.50

to 0.50, corresponding to the proposed steady flow value. As seen in Figure 5.56(b), the resulting peak was 1.5 percent low compared to 1.3 percent for the results for ω = 0.25.

5.3.2.4 Non-Linear Perturbation - Froude Number = 0.5

Once a comparison between the linear and non-linear case was examined, it was desirable to assess the performance of the various schemes for the case where the ambient flow was in motion. Therefore, a discharge of 10.25 cms (flow to the right), corresponding to a Froude number of 0.5, was superimposed on the previous test. The initial conditions specified are illustrated in Figure 5.58. The implicit schemes were run at time step increments of 0.4317 s and 0.2159 s, corresponding to Courant numbers of 1.0 and 0.5, respectively. The TG scheme was run at C = 0.5 and C = 0.25 (Δt = 0.1079 s). With the exception of the BFD scheme which was run at θ = 0.6, all implicit schemes were run for θ = 0.5. Results were presented at t = 12.95 s, after the progressive disturbance had travelled exactly 150 m. The peaks, each of depth 4.70 m were located at 140.89 and 430 m respectively. The results for the numerical tests are summarized in Table 5.6, while Table 5.7 provides the maximum negative depth deviations.

Figure 5.59 shows the results obtained using the BFD scheme. As expected from the linear stability analysis (Figures 4.4 and 4.5), the results for C = 1.0 illustrate that the peak phase speed was exact for the progressive wave, though the solution was quite dissipative compared to the regressive wave. However, the phase error associated with the regressive wave lead to leading disturbances, as seen in Figure 5.59 and Table 5.7. Again at C = 0.5 the peak was less damped, particularly the progressive wave. However, the shorter wavelengths were propagated too fast, resulting in leading disturbances on both waves. These results illustrate the difficulty in determining an optimum Courant number as either phase or amplitude accuracy must be sacrificed in the selection.

Table 5.6 Wave peaks for the non-linear propagation test for Froude number = 0.5 (semi-implicit).

Regressive Waves

C	Value	BFD*	BG	TG†	TG (lumped)†	DG††	DG†††	CDG††	CDG††††	LS
1	H	4.55906	4.63982			4.64282	4.58866	4.64024	4.62907	4.54772
	x	140	145			145	140	145	145	145
0.5	H	4.57761	4.64504	4.64089	4.59552	4.63961		4.63625		4.56453
	x	140	145	140	150	145		145		145
0.25	H			4.63975	4.59923					
	x			145	150					

Progressive Waves

1	H	4.46085	4.60325			4.59224	4.59251	4.59224	4.59597	4.54291
	x	425	420			425	425	425	425	425
0.5	H	4.50267	4.65740	4.59909	4.59563	4.62577		4.62576		4.56295
	x	430	425	430	425	425		425		425
0.25	H			4.63335	4.60115					
	x			430	425					

* Theta = 0.6 for BFD

† explicit runs

†† upwinding coefficient = 0.25

††† upwinding coefficient = $0.5 | U + c |$

†††† upwinding coefficient = 0.50

Table 5.7 Maximum negative depth deviations for the non-linear propagation test for Froude number = 0.5 (semi-implicit).

C	BFD*	BG	TG†	TG (lumped)†	DG††	DG†††	CDG††	CDG††††	LS
1	-0.05582	-0.07958			-0.04619	-0.03056	-0.04609	-0.03694	-0.02644
0.5	-0.07104	-0.03039	-0.04416	-0.05361	-0.02342		-0.02332		-0.02769
0.25			-0.01915	-0.06015					

* Theta = 0.6 for BFD

† explicit runs

†† upwinding coefficient = 0.25

††† upwinding coefficient = $0.5 \mid U + c \mid$

†††† upwinding coefficient = 0.50

The results for the BG scheme can be seen in Figure 5.60. The linear stability analysis predicted that no algorithmic damping would occur, though shorter wavelengths would be propagated too slowly (Figures 4.7 and 4.8). This can be seen in Figure 5.60 (a), for C = 1.0 where the trailing disturbances were accumulating behind the progressive wave in particular. Similar results were obtained for C = 0.5 as seen in part (b) of Figure 5.60, although the maximum negative depth deviation (Table 5.7) was lower.

Results for the TG scheme, run at Courant numbers of 0.5 and 0.25, are shown in Figure 5.61. As was expected from the linear stability analysis, the progressive wave experienced some leading oscillations due to phase error, particularly at C = 0.5. For the regressive wave, the phase accuracy at C = 0.5 was considerably improved, although for both Courant numbers this scheme was somewhat less dissipative of the short wavelengths (Figure 4.14). Consequently, some trailing disturbances persisted behind the regressive wave, particularly at C = 0.25. Results for the lumped TG are shown in Figure 5.62, where the trailing phase errors were much worse, particularly at C = 0.25 as Table 5.7 shows.

The results for the DG scheme are shown in Figure 5.63 for an $\omega = 0.25$. As expected from the linear stability analysis (Figure 4.16), very little damping of the peaks occurred for the progressive wave. However, some trailing disturbances were noted, particularly at a Courant number of 1.0. For the regressive wave, the phase accuracy was improved at C = 1.0, as supported by the linear stability analysis (Figure 4.17). Similar results obtained for the CDG scheme, as seen in Figure 5.64 and Table 5.6, although in this case the phase accuracy was improved at both Courant numbers. These results also agree with the predictions of the linear stability analysis (Figures 4.22 and 4.23).

Once again, to examine the effect of the increased upwinding coefficient used in the steady flow analysis, the DG was next run using a value of $\omega = 0.50\,|U+c|$. Table 5.6 and Figure 5.65(a), show a slight decrease in the peaks, associated with a leading phase error. This was expected from the linear stability analysis (Figures 4.39 and 4.40). Similarly, the CDG scheme was run at a value of ω increased to 0.50, corresponding to the proposed steady flow value. As seen in Figure 5.65(b) and Table 5.6, the resulting solution was close to the results for $\omega = 0.25$, although disturbances were somewhat increased behind the progressive wave.

Figure 5.72 illustrates the results for the LS method. Although the phase accuracy at C = 1.0 was improved for the regressive wave, this scheme was significantly less dissipative of $2\Delta x$ oscillations (Figures 4.30 and 4.31). Therefore, trailing disturbance were observed behind the regressive wave. At C = 0.5, the scheme was also less dissipative of $2\Delta x$ disturbances for the regressive wave than the progressive wave. Furthermore, the phase accuracy was reduced as well. Consequently, trailing disturbance were observed behind the regressive wave here as well.

5.3.2.5 Non-Linear Perturbation - Froude Number = 1.0

For this test, a discharge of 20.51 cms, corresponding to a Froude number of 1.0, was superimposed on the original non-linear test. The initial conditions specified are illustrated in Figure 5.67. The implicit schemes were run at time step increments of 0.3670 s and 0.1835 s, corresponding to Courant numbers of approximately 1.0 and 0.5, respectively. The TG scheme was run at $C \approx 0.5$ and $C \approx 0.25$ ($\Delta t = 0.09175$ s). With the exception of the BFD scheme which was run at $\theta = 0.6$, all implicit schemes were run for $\theta = 0.5$. Results were presented at t = 11.0 s, again after the progressive disturbance had travelled 159.8 m.

Because this test represented a critical flow, it was run with both the subcritical and supercritical BFD algorithms to determine which, if either, would be applicable. As it turned out, only the subcritical algorithm was successful in achieving a stable solution. The results are illustrated in Figure 5.68, where noticeable oscillations in the solution, particularly at C = 0.5 were observed. This was expected from the linear stability analysis for the regressive wave (Figure 4.6) as this scheme was only slightly dissipative of 2Δx disturbances and not dissipative at all for other wavelengths. It is interesting to note that both leading and trailing disturbances were observed in the solution, although the linear stability analysis predicted only leading phase error. This might be attributed to the non-linearity or the fact that the stability analysis was run at a **F** = 0.99 rather than 1.0, to avoid a singularity in the calculation.

For the BG scheme, the linear stability analysis predicted that the regressive wave would also be non-dissipative although the phase accuracy would be improved, particularly at C = 0.5 (Figure 4.9). This was evident in the results for the BG method, shown in Figure 5.69. Similar effects were noted in Figure 5.70 for the TG scheme where the linear stability analysis had predicted both perfect phase accuracy and no dissipation at any Courant number (Figure 4.15). Tendencies were similar for the lumped TG, as seen in Figure 5.71, but again results were somewhat worse than for the consistent TG.

The linear stability analysis had predicted that the DG scheme would be entirely non-dissipative for the regressive wave at **F** = 0.99. In addition, the phase accuracy was not significantly altered from that for the progressive wave (Figures 4.16 and 4.18). This was evident in Figure 5.72 for the DG at ω = 0.25, where in contrast to the progressive wave, the regressive wave matched the exact solution almost perfectly and no trailing disturbances were observed. The DG and CDG produced identical results at F = 1.0.

Again, the DG was run using a value of $\omega = 0.50\ |U+c|$ to examine the effect of the increased upwinding coefficient used in the steady flow analysis. The results, shown in Figure 5.73 (a), showed an improvement for the progressive wave (0.0216 m higher) but a slight decrease in the regressive wave peak (0.03193 m lower). The CDG scheme, run at a value of ω increased to 0.50, corresponding to the proposed steady flow value, exhibited only a slight decrease in the regressive wave (0.00873 m lower). Disturbances were somewhat increased behind the progressive wave, as seen in Figure 5.73 (b), and its peak was slightly (0.00360 m) higher.

Figure 5.74 illustrates the results for the LS method. Although the linear stability analysis predicted that phase accuracy at C = 1.0 would be improved for the regressive wave, that for C = 0.5 would be somewhat reduced (Figure 4.32). In addition this scheme was expected to be almost entirely non-dissipative for the regressive wave. As Figure 5.74 (a) illustrates, trailing disturbance occurring behind the regressive wave affected the progressive wave such that the solution failed shortly after t = 1.468 s. At C = 0.5, Figure 5.74 (b), an excellent solution was achieved although some $2\Delta x$ oscillations were noted behind the regressive wave.

5.3.2.6 Non-Linear Perturbation - Froude Number = 2.0

For this test, a discharge of 41.02 cms, corresponding to a Froude number of 2.0, was superimposed on the original non-linear test. The initial conditions specified are illustrated in Figure 5.75. The implicit schemes were run at time step increments of 0.2566 s and 0.1283 s, corresponding to Courant numbers of approximately 1.0 and 0.5, respectively. The TG scheme was run at $C \approx 0.5$ and $C \approx 0.25$ ($\Delta t = 0.06415$ s). With the exception of the BFD scheme which was run at $\theta = 0.6$, all implicit schemes were run for $\theta = 0.5$. Results were presented a t = 7.7 s, after the progressive disturbance had travelled

156.8 m. It should be remembered that in this test, both the regressive and progressive waves moved with the flow.

Figure 5.76 shows the results obtained using the BFD scheme. As expected from the linear stability analysis (Figures 4.4 and 4.5), the results for C = 1.0 illustrate that although the peak phase speed was good for the progressive wave, the peak was quite damped. For the regressive wave, results were significantly less dissipative and only minor leading disturbances were observed. Again at C = 0.5 the peak was less damped, particularly the progressive wave, though leading disturbances were observed on both waves.

The results for the BG scheme can be seen in Figure 5.77. The linear stability analysis predicted that no algorithmic damping would occur, though shorter wavelengths would be propagated too slowly (Figures 4.7 and 4.8). This can be seen in Figure 5.77 (a), for C = 1.0 where the trailing disturbances accumulating behind the progressive wave actually caused the solution to fail shortly after t = 5.13 s. On the other hand, for C = 0.5 a successful result was obtained.

Results for the TG scheme are shown in Figure 5.78. Compared to the progressive wave, the phase accuracy for the regressive wave at C = 0.5 was improved. From the linear stability analysis this scheme was expected to somewhat less dissipative of the short wavelengths in the regressive wave than the progressive wave for both Courant numbers (Figures 4.13 and 4.14). Consequently, some trailing disturbances persisted behind the regressive wave, particularly at C = 0.25. Results for the lumped TG are shown in Figure 5.79, where the trailing phase errors were much worse at both C = 0.25 and 0.50 than for the consistent TG.

The results for the DG scheme are shown in Figure 5.80 for ω = 0.25. Compared with the progressive wave, the phase accuracy was improved for the regressive wave at

C = 1.0, as supported by the linear stability analysis (Figures 4.16 and 4.17). Similar results were obtained for the CDG scheme, as seen in Figure 5.81.

Figure 5.82 (a) shows the effects of increasing the upwinding coefficient to a value of $\omega = 0.50\,|U+c|$ for the DG scheme. A slight decrease (0.01503 m) in the peak associated with a leading phase error was observed for the regressive wave. However, a slight increase in the progressive wave peak was observed (0.03397 m). Similarly the results for the CDG scheme, run with the value of ω increased to 0.50, are presented in Figure 5.82(b). Again, the resulting solution was nearly identical (within 0.006 m) to the results for ω = 0.25, although disturbances were still apparent behind the progressive wave.

Figure 5.83 illustrates the results for the LS method. Again, the linear stability analysis had predicted the phase accuracy at C = 1.0 would be improved for the regressive wave. However, this scheme was significantly less dissipative of $2\Delta x$ disturbances (Figures 4.30 and 4.31). Consequently, as Figure 5.90 (a) illustrates, trailing oscillations associated with the regressive wave caused the solution to fail shortly after t = 3.34 s. Although results were somewhat better at this point for C = 0.5, this solution failed as well.

5.3.2.7 Discussion of Results

These tests clearly illustrate the need to compromise between the phase and amplitude accuracy when using the BFD scheme. In contrast, the finite element schemes, particularly the DG and CDG methods displayed both good phase and amplitude accuracy, at C = 0.5 in particular, for all runs. In addition, it appeared that the TG scheme was the best near critical flow, exhibiting perfect phase accuracy and good peak preservation for the regressive wave, as predicted by the linear stability analysis. However, the additional boundary condition requirements of this scheme must be kept in mind. Finally, it would appear that the LS method is unsuited to critical and supercritical flows, particularly for a Courant number in excess of 0.5. This conclusion is supported by the steady flow tests discussed above.

5.3.3 Dam Break Analysis

5.3.3.1 General

This test, reproduced from Fennema *et al* (1987), simulates the instantaneous failure of a dam in a horizontal, frictionless channel. In this case, the channel geometry consisted of a unit width section. Eighty elements, of length 25 m were used in the analysis, for a total of 81 nodes. The dam itself was approximated over a single, 25 m long element. The discharge was initially set to zero at all nodes and through the upstream half of the domain the initial flow depth, H_o, was set to 10 m. For the downstream half two flow depths, H_1, were studied. First a depth of 5 m ($H_1/H_o = 0.5$) was tested and then 0.5 m ($H_1/H_o = 0.05$). The boundary conditions provided at both ends specified a discharge of zero.

It should be noted that, for the Petrov-Galerkin DG and CDG schemes, best results for these tests were obtained by updating the upwinding matrix [W] every iteration rather than every time step, as was done in all the previous test. In fact, for the $H_1/H_o = 0.05$ test, a stable solution could not be achieved unless this was done. Furthermore, convergence for this test required the use of only the [K'] matrix for approximating the Jacobian, that is without including any extra terms.

5.3.3.2 Dam Break - $H_1/H_o = 0.5$

Figure 5.84 illustrates the initial depth profile provided for this test. To achieve optimal solutions, the implicit finite element schemes were run at a time step increment of 1.25 s. This corresponded to a Courant number of 0.47, based on the speed of the progressive surge (which was determined, analytically, to be 9.35 m/s). The TG scheme was run at a time step increment of 0.625 s corresponding to a Courant number of 0.23. All implicit finite element schemes were run for $\theta = 0.5$ and $\theta = 1.0$. For comparison, the BFD scheme was run at $\theta = 0.6$ using a time step increment of 2.50 s, corresponding to a Courant number of 0.94, as well as for $\Delta t = 1.25$s. As in the original test, results were presented after 60s. Details of the maximum depth deviations associated with the progressive wave are provided in Table 5.8.

Figure 5.85 shows the results for the BFD scheme and, as expected from the linear stability analysis, leading disturbances were observed in the solution for $C \approx 0.5$ as seen in in part (a). The surge itself was spread over 3 elements and only negligible oscillations were observed ahead of the regressive wave. For the run conducted at $C \approx 1.0$, shown in part (b), noticeably fewer oscillations were present, again as predicted by the linear stability analysis, while in this case, the shock was spread over two elements.

Results for the BG method are shown in Figure 5.86. As expected, severe oscillations were observed, particularly trailing the shock, in the solution for $\theta = 0.5$ shown in part (a). When the implicitness was increased to 1.0, these disturbances were effectively damped at the expense of smearing the surge over 5 elements.

Figure 5.87 illustrates the results obtained for the TG scheme, run at $C \approx 0.25$. Although the surge was distributed over just two elements, oscillations just upstream and downstream of the jump were observed. This scheme was somewhat less dissipative of the regressive wave than the BFD run.

Results for the DG scheme, run at $\omega = 0.25$, are shown in Figure 5.88. As seen in in part (a) for $\theta = 0.5$, the shock was distributed over two elements as in the TG scheme, though leading and trailing disturbances were noticeably smaller. It is interesting to note, however, that the regressive wave was subject to oscillations and trailing waves. Results

Table 5.8 Maximum negative progressive wave depth deviations for the dambreak test (H1/Ho = 0.5).

Δt	Theta	BFD	BG	TG	DG††	CDG††	LS
2.5	1						
	0.6	-0.092					
	0.5						
1.25	1		-0.124		0.000	0.000	-0.092
	0.6	-0.680					
	0.5		-1.476		-0.234	-0.194	-0.299
	0.5						
0.625	0			-0.402			

†† upwinding coefficient = 0.25

were considerably smoother for the case where $\theta = 1.0$ was used, though like the BG scheme, this was at the expense of smearing the surge, as well as the regressive wave.

For the CDG scheme, which was also run at $\omega = 0.25$, the shock was again distributed over only two elements, as seen in Figure 5.89 (a). However, in this case, the localized disturbances were somewhat smaller than those observed in the corresponding DG test. Furthermore, the regressive wave was completely free of the oscillations observed in that case. The results for $\theta = 1.0$, shown in Figure 5.89 (b) were nearly identical to the DG test results.

Finally, the results for the LS scheme are presented in Figure 5.90. The extremely dissipative nature of this scheme is evident both for the implicit and semi-implicit runs. In addition, a disturbance originating at the dam location persisted throughout the simulation period.

5.3.3.3 Dam Break - $H_1/H_0 = 0.05$

Figure 5.91 illustrates the initial depth profile provided for this test. This case presented a much more difficult modeling situation, as the limited downstream flow depth made this test particularly susceptible to failure due to depth oscillations. The implicit finite element schemes were run at a time step increments of 0.625 s corresponding to a Courant number of 0.26. This was again based on the speed of the progressive surge which was determined, analytically, to be 10.47 m/s. The TG scheme was run at a time step increment of 0.3125 s corresponding to a Courant number of 0.13, as well as at $\Delta t = 0.625$ s. Once again, the implicit finite element schemes were run for $\theta = 0.5$ and $\theta = 1.0$. The BFD scheme was not successful in providing a comparative solution for this test. As in the original test, results were presented after 60s.

Results for the BG method are shown in Figure 5.92. Severe oscillations were observed, particularly trailing the shock, in the solution for $\theta = 0.5$ shown in part (a). When the implicitness was increased to 1.0, these disturbances were effectively, but not completely, damped particularly for the regressive wave. The surge front was much sharper for the fully implicit run in this case, compared with the $H_1/H_o = 0.5$ test results.

Figure 5.93 illustrates the results obtained for the TG scheme. Good results were obtained using at $C = 0.13$ as seen in part (a), representing the shock over a single element. However, oscillations were observed upstream of the surge. For the $C = 0.26$ run, shown in part (b), a stable solution could not be obtained and the solution failed shortly after $t = 10$s. This was due to the leading disturbances which deteriorated into negative depths.

Results for the DG scheme, run at $\omega = 0.25$, are shown in Figure 5.94. As seen in in part (a) for $\theta = 0.5$, the shock was distributed over two elements, though leading and trailing disturbances were observed. The regressive wave exhibited a disturbance as well. This solution failed just after $t = 55.625$s, and in fact a stable solution could not be obtained for shorter time step increments either. Results were considerably worse when $\theta = 1.0$ was used, as seen in in part (b) where the solution failed just after 17.5s. Again, a stable solution could not be obtained by attempting shorter time step increments. Increasing the magnitude of the upwinding was not successful in achieving a stable solution for this method either. Again, as in the TG test leading phase error, leading to negative depths, was the cause of the solution failure.

For the CDG scheme, which was also run at $\omega = 0.25$, the shock was distributed over two elements for $\theta = 0.5$, as seen in Figure 5.95 (a). Again localized disturbances were observed in the vicinity of the shock. The results for $\theta = 1.0$, shown in Figure 5.95 (b) are much better than the corresponding $H_1/H_o = 0.5$ test results, though the regressive wave was somewhat dampened and the surge was spread over two elements. As for the two previous methods, leading phase error made this method susceptible to failure from negative depths ahead of the surge. Runs done at larger time step increments quickly failed.

The results for the LS scheme, shown in Figure 5.96 describe the solution obtained just before the solution "blew up". For the semi-implicit run, this occurred just after t = 5.625s. In the case of the fully implicit test, where the numerical dissipation would have been doubled, the solution in fact failed sooner - just after t = 1.875s. As in the DG scheme, shorter time steps were no more stable failing by the same negative downstream depth problem as the other methods.

5.3.3.4 Discussion of Results

These dam failure simulations provided a fairly rigorous test of the various methods by combining progressive and regressive disturbance propagation with a shock situation. The second problem in particular proved to be a method "breaker" by creating the possibility for negative depths when oscillations occurred. In general, the TG and CDG scheme performed the best for this test, while again the LS scheme was the most unstable.

At this point the CDG scheme had clearly emerged as the most promising method. However, except for the poor results obtained for the $H_1/H_o = 0.05$ dam break test the DG scheme appeared very good as well, particularly when the upwinding coefficient, ω, was increased. Because such an increase had only caused marginal consequences in the propagation of the disturbances discussed in Section 5.3.2, a more stringent test was designed to assess this effect. This is discussed next.

5.3.4 Propagation of a Wave Through an Hydraulic Jump

5.3.4.1 Introduction

This test was designed to combine the disturbance propagation problem with the hydraulic jump situation tested earlier. Practically speaking, this might occur when routing a flood wave through a long channel involving both mild and steep reaches, such as the Hay River case to be discussed in Chapter 6. Fifty-nine elements, spaced at 5 m intervals, were used in the analysis for a total of 60 nodes. As in the original hydraulic jump test, the base discharge was 5 cms, the initial supercritical flow depth was 1.008 m and the downstream subcritical flow depth was 1.80 m. In addition, a disturbance 1.0 m high spread over 10 elements was superimposed just upstream of the jump. The initial conditions specified for this test are illustrated in Figure 5.97. The boundary conditions provided were area and discharge at the upstream end, and area at the downstream end.

This test could not be run for the BFD scheme due to the combination of subcritical and supercritical flow. For the implicit schemes: BG, DG, CDG and LS, the test was run semi-implicit at a Δt of 0.2086s, corresponding to a Courant number of 0.5 based on the initial value of |U+c| at the peak which was 11.985 m/s. The explicit Taylor-Galerkin scheme was run at a Δt of 0.1043s, corresponding to a Courant number of 0.25. The tests were run for a total time of 25.03 s, allowing the wave to pass well downstream of the jump.

5.3.4.2 Discussion of Results

Figure 5.98 show the results for the BG scheme, after only 50 time steps, at t = 10.43 s. Oscillations in the supercritical reach were substantial and the solution failed just after the wave passed through the jump. In contrast the TG scheme, as seen in Figure 5.99, was successful in propagating the wave through the jump. However, severe oscillations were observed throughout the entire supercritical flow reach as well as through most of the subcritical flow upstream of the wave.

As illustrated in Figure 5.100, the results obtained for the DG scheme, when run at $\omega = 0.25$, showed tendencies similar the TG method, with severe oscillations in the vicinity of the original jump. As shown in Figure 5.101, increasing the upwinding coefficient by a factor of $|U+c|$ was successful in suppressing these oscillations and preserving the jump face. However, the peak flow depth and discharge of the propagated wave were reduced by about 15 and 25%, respectively. In addition, severe oscillations were observed ahead of this wave for both the depth and discharge.

The results for the CDG method, run at $\omega = 0.25$, are shown in Figure 5.102. In contrast to the DG runs, this scheme preserved the jump well, with only minor localized oscillations while maintaining the peak of the propagated wave. In essence the CDG scheme has the ability, therefore, to exhibit the good propagation behavior of the standard DG ($\omega = 0.25$), without requiring any additional upwinding.

Results for the LS scheme are presented in Figure 5.103. The LS scheme was successful in propagating the wave through the jump although oscillations, particularly in the flow depth, were observed upstream of the jump. In addition, both the peak depth and discharge were somewhat reduced compared with the results of the CDG and TG tests. However, this scheme was not as dissipative as the DG tested with the upwinding increased.

5.3.5 Diffusive and Kinematic Waves

5.3.5.1 Introduction

Based on these results, the CDG scheme clearly emerges as superior in handling unsteady flow in horizontal, frictionless channels. However, the question still remained as to its suitability to the other extreme. That is friction dominated flows, particularly kinematic waves, given the results of the linear stability analysis for these problems. For these tests the definition of β, was based on the following:

$$\beta = \frac{2\Delta x}{C^2 H} \qquad [5.1]$$

with H defined as the initial depth at the peak. In all cases, the uniform flow discharge was used as the boundary condition at both boundaries.

5.3.5.2 Results for $\beta = 5.5$

Figure 5.104 illustrates the initial conditions used for this test. A10 m wide channel on a mild slope of 0.005 was used. One hundred nodes, spaced at 1 km intervals were used to define the channel. The flow was initially set up as uniform with a discharge of 31.37 cms and an effective roughness of 5 m. The various methods were then tested to confirm a uniform flow depth of 2.86 m. The flow depth was increased in 2 m increments over 5 km starting at the second node, and then reduced in the same fashion to produce a symmetrical disturbance 10 km long, with a peak flow depth of 12.5 m. The corresponding initial discharge was calculated based on a uniform flow assumption, with a maximum of about 280 cms occurring at the peak. Thus the value of β was 5.5.

For the implicit finite element schemes, the tests were run at $\theta = 0.5$ and a time step increment of 180 s corresponding to a Courant number of 0.57, based on the propagation speed of the peak. The LS and the explicit TG scheme had to be run at $\Delta t = 15$s, corresponding to a Courant number of 0.05 to achieve a stable solution, while the BFD was run at $\Delta t = 360$ s ($C = 1.1$) and $\theta = 0.6$.

Figure 5.105 shows the results obtained after at 2, 4, and 6 hours for the BFD and BG schemes, Figure 5.106 the DG and CDG schemes (both run using $\omega = 0.25$) and

Figure 5.107 the TG and LS method. A summary of the peaks obtained at 2, 4, and 6 hours are provided in Table 5.9. As the table illustrates,results for the BFD and LS schemes were more dissipated than for the other methods. In addition, the LS scheme experienced some oscillation at the downstream boundary which occurred right at the beginning of the simulation. Based on these results, any of the BG, TG, DG or CDG methods could be expected to produce nearly identical results to the others for this β.

These peaks propagated at an almost constant speed of 3.15 m/s, roughly 6% slower than the kinematic wave speed for this test. Given that the peak discharges continuously decreased throughout the simulation, it is reasonable to assume that the wave was diffusive rather than kinematic. Therefore, as expected from the linear stability analysis, no stability problems were encountered for the DG or CDG schemes.

5.3.5.3 Results for β = 61

For this test, the effective roughness was increased to 10 m and the node spacing to 5 km. This reduced the slope to 0.001 and the discharge to 7.42 cms for the same uniform flow depth. Figure 5.108 illustrates the initial conditions. The flow depth was increased in 2.0 m increments over 25 km starting at the second node, and then reduced in the same fashion to produce a symmetrical disturbance 50 km long, with a peak flow depth of 12.5 m. The corresponding initial discharge was calculated based on a uniform flow assumption, with a maximum of about 85 cms occurring at the peak. Thus the value of β was 61.

For the implicit finite element schemes, the tests were run at $\theta = 0.5$ and a time step increment of 1800 s corresponding to a Courant number of about 0.26. The explicit TG scheme was run at half this increment, corresponding to a Courant number of 0.13. The BFD was run at the same value ($C \approx 0.26$) and $\theta = 0.6$. The TG and LS schemes could not provide stable solutions at any time step increment.

Table 5.9 Wave peaks for the diffusive wave test, ß = 5.5 (semi-implicit).

t (hrs)	Value	BFD*	BG	TG†	DG††	CDG††	LS
2	H	6.81172	7.03953	7.05224	7.03286	7.03164	6.74103
	Q	123.818	129.954	130.305	129.707	129.672	122.023
4	H	5.95638	6.17721	6.20677	6.16917	6.16845	5.99623
	Q	101.902	107.354	108.395	107.194	107.173	102.942
6	H	5.51365	5.71984	5.76952	5.71326	5.71273	5.59659
	Q	90.787	95.931	97.142	95.772	95.761	92.658

* Theta = 0.6 for BFD

† explicit runs

†† upwinding coefficient = 0.25

Figure 5.109 shows the results obtained after 2,4 and 6 hours for the BFD and BG schemes and Figure 5.110 the DG and CDG schemes (both run using $\omega = 0.25$). Peaks at 5 and 10 hours are also presented in Table 5.10. Again the BFD was more dissipative than the finite element schemes, and experienced a leading disturbance associated with the phase error. The peak propagated at an almost constant speed of 0.73 m/s, equal to the kinematic wave speed for this test.

5.3.5.4 Results for $\beta = 117$

For this test, the effective roughness was increased to a somewhat unrealistic 15 m. The initial condition were otherwise the same as for $\beta = 61$. The corresponding initial discharge was decreased to 3.6 cms, with a maximum of about 61 cms occurring at the peak. Thus the value of β was 117.

Only the BFD scheme was successful in producing a stable solution for this test, indicating that the maximum allowable β for the DG and CDG schemes would be somewhere between 61 and 117. Results at 5 and 10 hours are shown in Figure 5.112.

5.3.6 Comparison of Methods

Based on the results of the steady and unsteady numerical analysis, it must be concluded that the DG and CDG Petrov-Galerkin schemes are clearly superior to both the other finite element schemes and the BFD scheme. Furthermore, in considering both the progressive and regressive waves in quantifying the upwind weighting, the CDG was able to achieve good results without increasing the upwinding coefficient, unlike the DG. Therefore, only the CDG scheme is considered in the ensuing discussion.

Table 5.10 Wave peaks for the diffusive wave test, beta = 61 (semi-implicit).

t (hrs)	**Value**	**BFD***	**BG**	**DG††**	**CDG††**
5	H	6.43557	6.52926	6.51199	6.51120
	Q	32.579	33.268	33.164	33.156
10	H	5.63613	5.72484	5.71526	5.71483
	Q	26.238	26.911	26.844	26.842

* Theta = 0.6 for BFD

†† upwinding coefficient = 0.25

5.4 SYMMETRIC AND NON-CONSERVATION FORMULATIONS FOR THE CDG SCHEME

5.4.1 Introduction

Because the application of the CDG method to symmetric systems of hyperbolic equations assures orthogonality of the eigenvectors of the upwinding matrix, the investigation of this scheme was extended to examine a symmetric form of the St. Venant equations. This allowed for the determination of whether or not the cross-upwinding effect was significant enough to require the use of symmetric forms in general shallow flow modeling. As this necessitated the use of a non-conservation system, a comparison of conservative and non-conservative non-symmetric formulations was also done. Details of these formulations are provided in Appendix B.

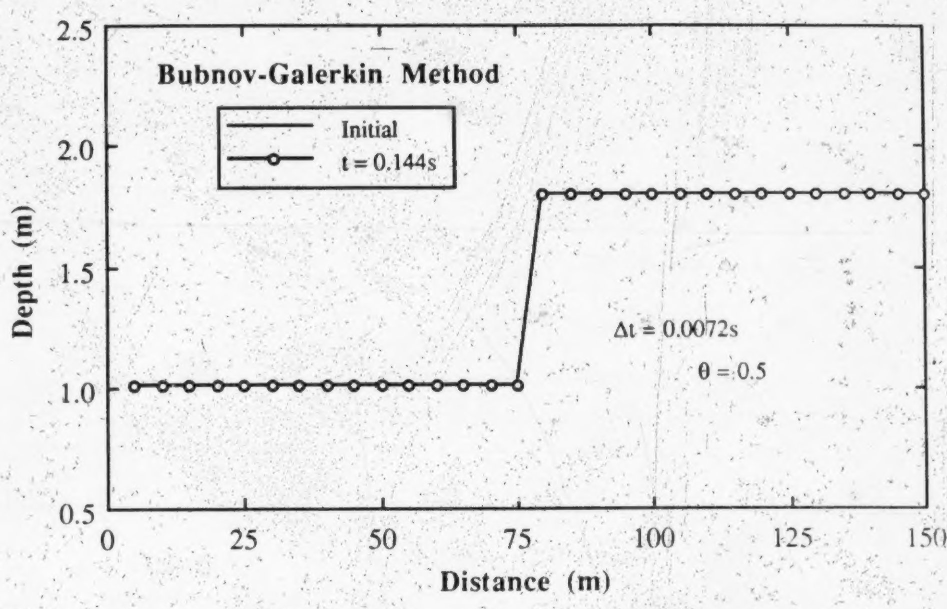
5.4.2 Discussion of Results

The first test recreated for this analysis was the non-linear perturbation problem for $F = 0.0$. Figure 5.113 illustrates the results obtained using the conservation form of the equations (as discussed above) compared with the symmetric formulation (a) and the non-conservation form (b). As Figure 5.113 (a) shows, there appeared to be very little cross-upwinding effect as the results for the symmetric formulation were only marginally higher than the non-symmetric conservation form. However, as seen in Figure 5.113 (b), a noticeable difference between the results for the conservation and non-conservation form of the equations was observed. As the symmetric formulation required the use of a non-conservation form of the equations, it might be concluded that the cross-upwinding effect was of comparable magnitude to the error in using the non-conservation form of the equations.

The next test considered was the non-linear perturbation problem for **F** = 0.5. As Figure 5.114 (a) shows, there appeared to be very little cross-upwinding effect as the results for the symmetric formulation were negligibly different from the non-symmetric conservation form. Furthermore, as seen in Figure 5.114 (b), a much less significant difference between the results for the non-conservation form of the equations was observed than when **F** = 0.0.

Finally, the dambreak problem for $H_1/H_o = 0.5$ was tested. Figure 5.115 (a) shows the comparison between the results for the symmetric and non-symmetric versions where it appears that the non-symmetric solution was slightly more encumbered by parasitic oscillations. In contrast to the two previous tests, Figure 5.115 (b) showed no difference whatsoever in comparing the solution for the conservation formulation and the non-conservation formulation.

Based on these results, it appears that the conservation form of the St. Venant equations are quite appropriate for handling one-dimensional flow problems. Cross-upwinding effects, if significant, could not be remedied thorough the use of a symmetric formulation without resorting to a non-conservation form, which appeared to counteract any beneficial effects afforded by the symmetric approach. In considering extensions to two-dimensional flow, it might be advisable to consider the symmetric formulation if a non-conservation form of the governing equations is used.



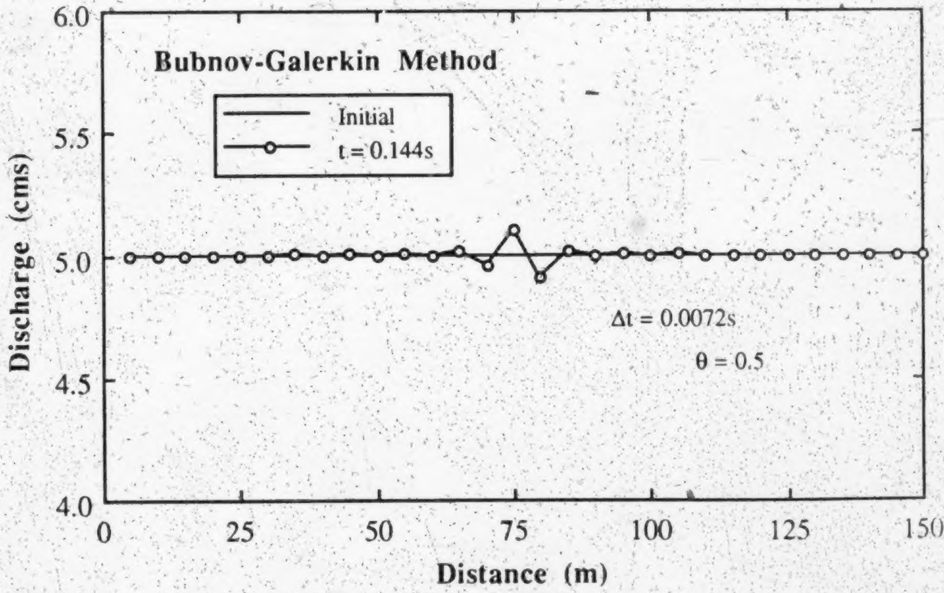


Figure 5.1 Hydraulic jump (t = 0.144s), Bubnov-Galerkin method, θ = 0.5, Δt = 0.0072s.

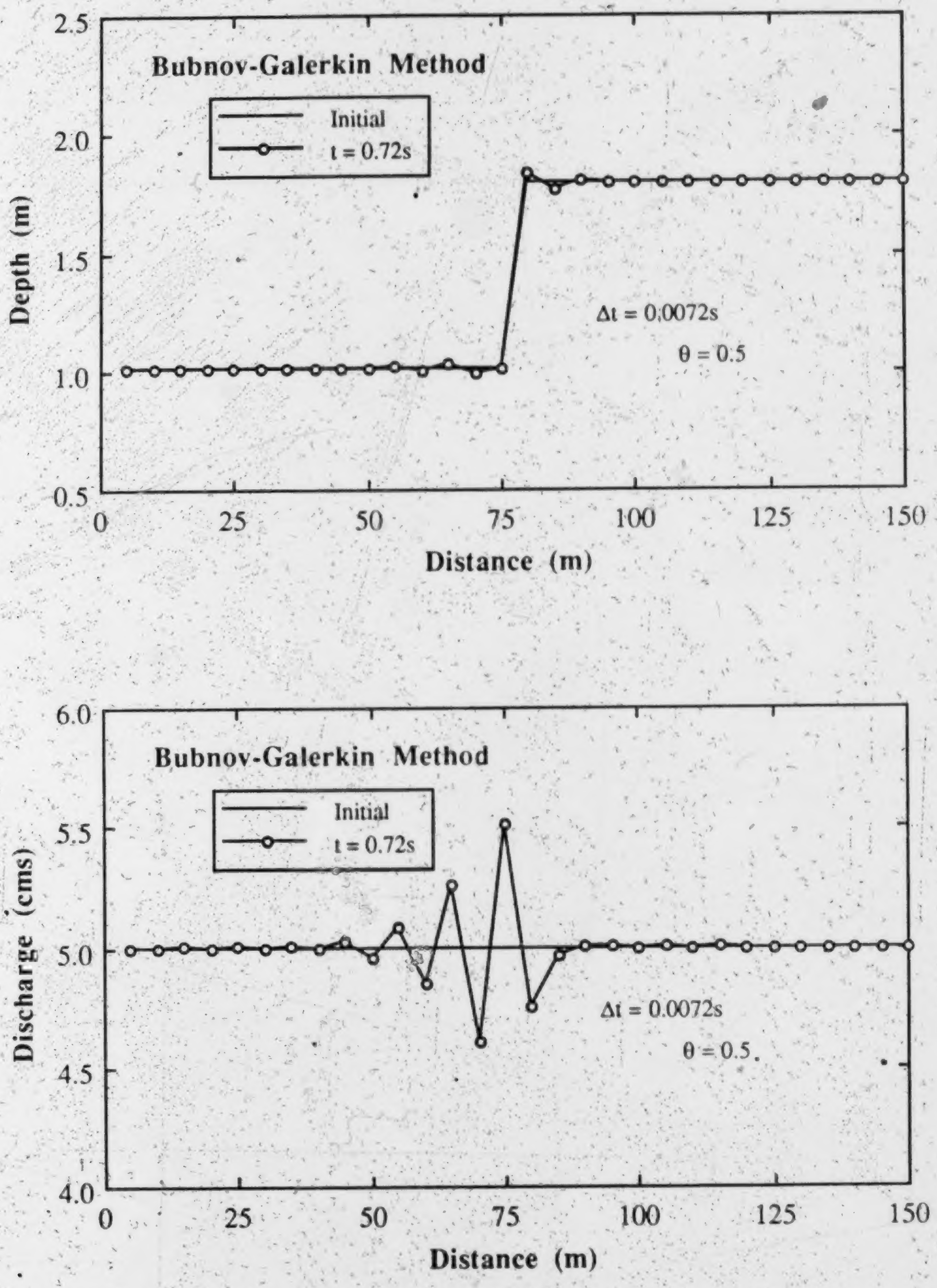
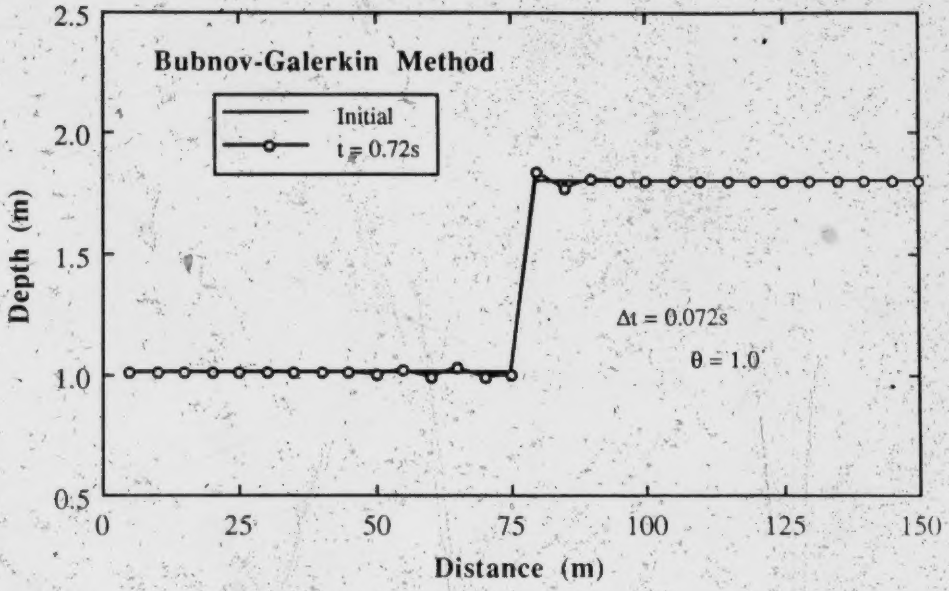


Figure 5.2 Hydraulic jump (t = 0.72s), Bubnov-Galerkin method, θ = 0.5, Δt = 0.0072s.



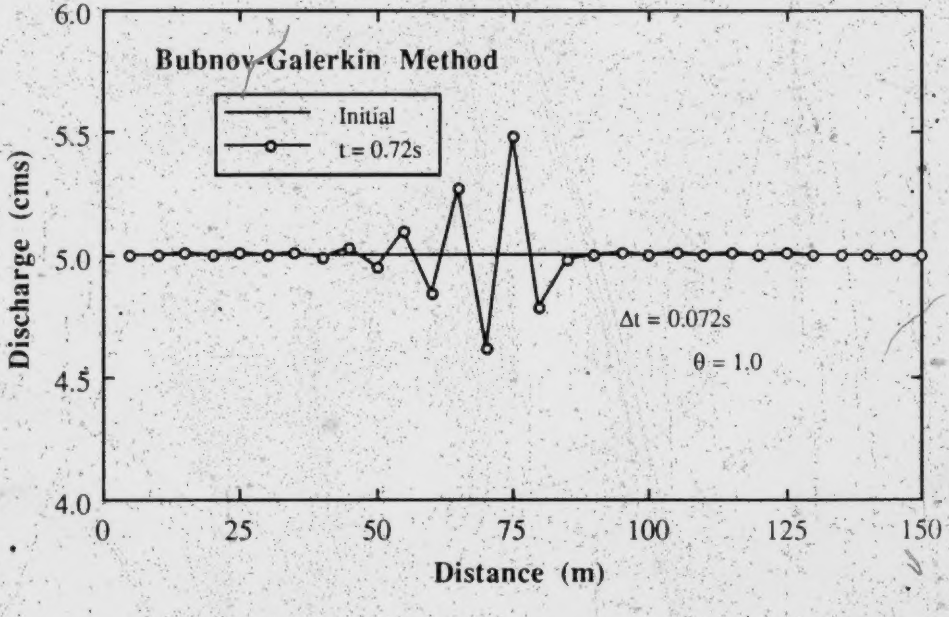
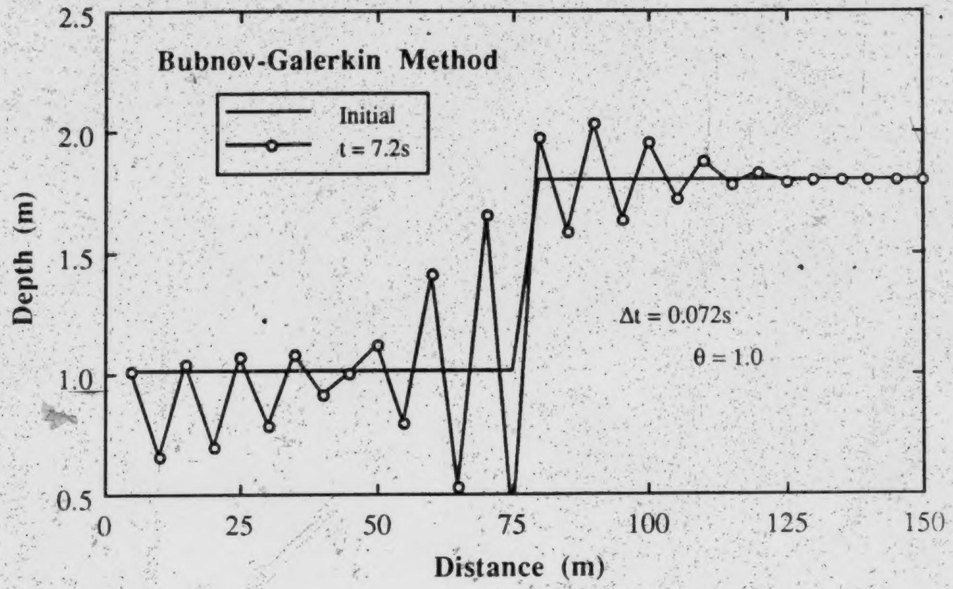


Figure 5.3 Hydraulic jump (t = 0.72s), Bubnov-Galerkin method, θ = 1.0, Δt = 0.072s.



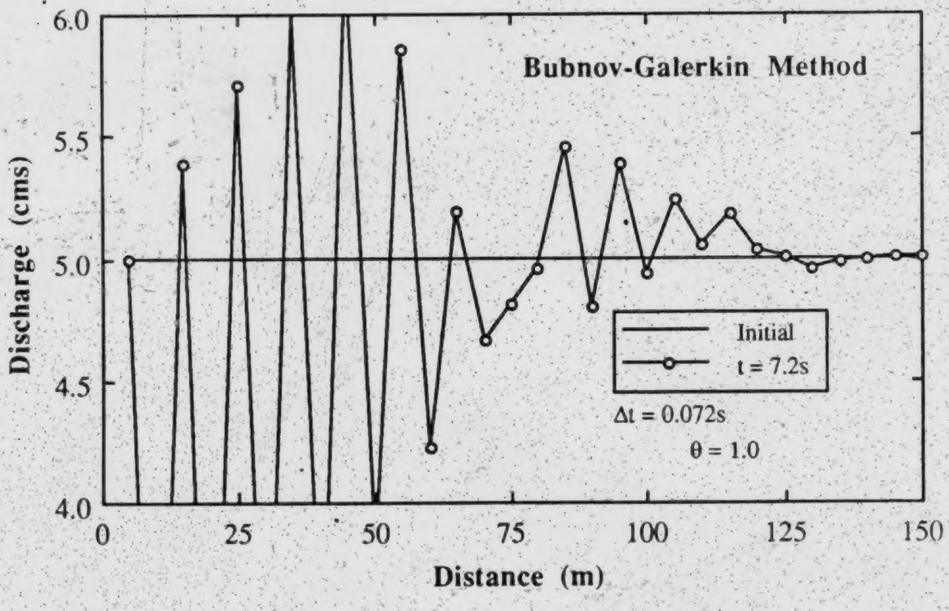
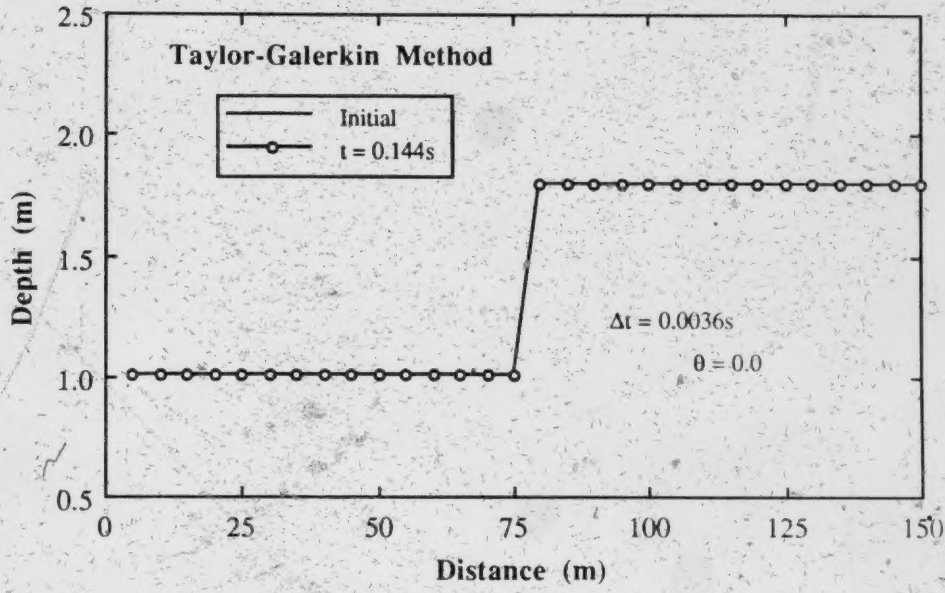


Figure 5.4 Hydraulic jump (t = 7.2s), Bubnov-Galerkin method, θ = 1.0, Δt = 0.072s.



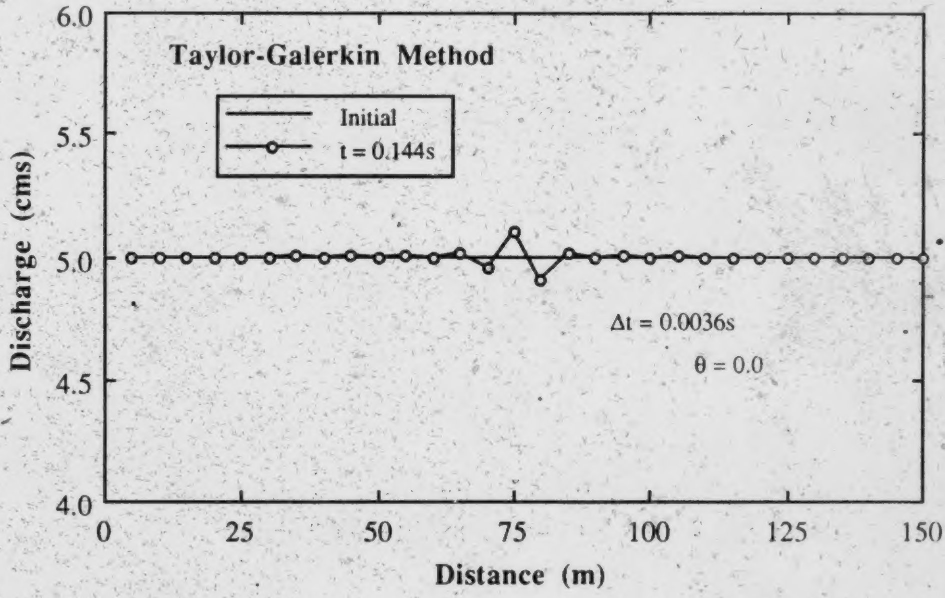
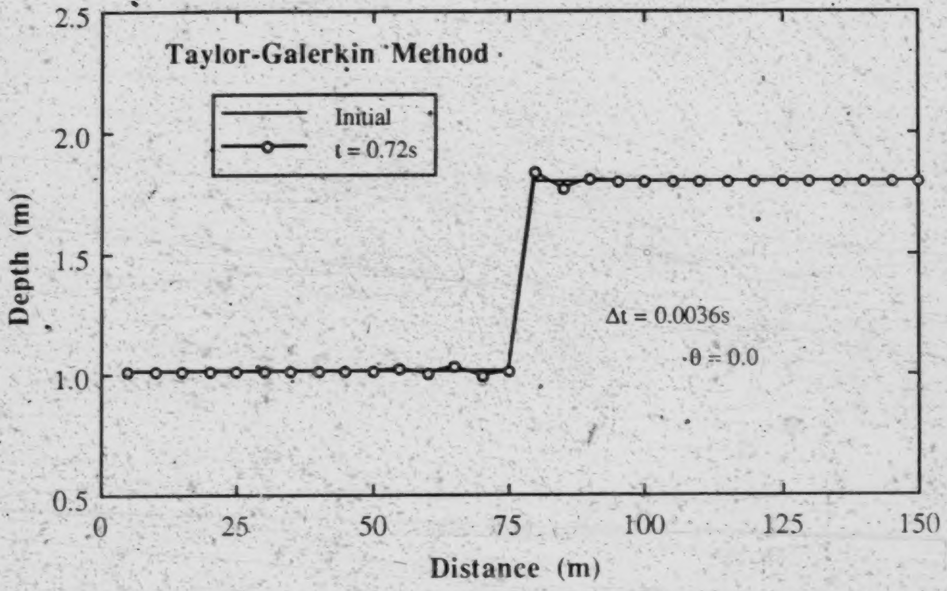


Figure 5.5 Hydraulic jump (t = 0.144s), Taylor-Galerkin method, θ = 0.0, Δt = 0.0036s.



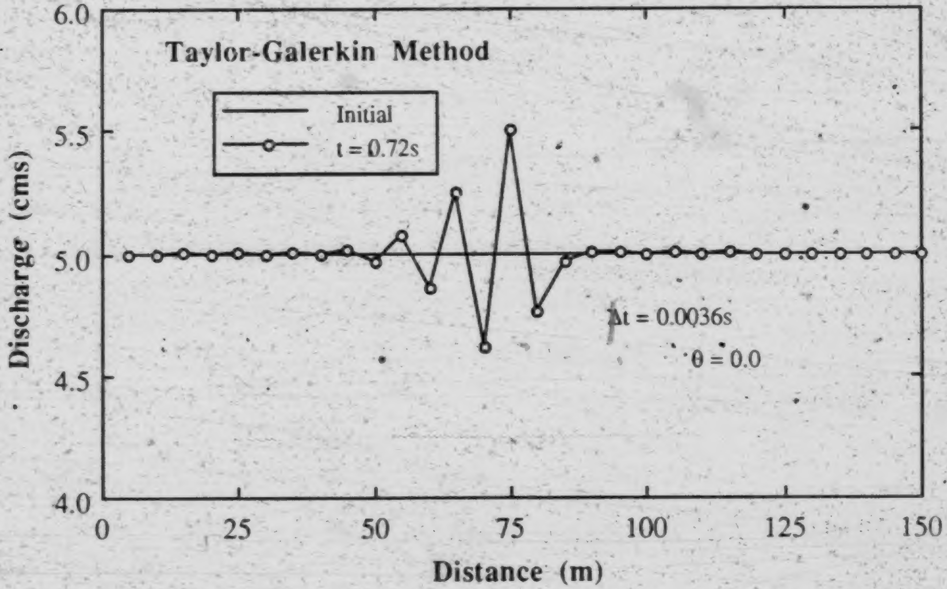
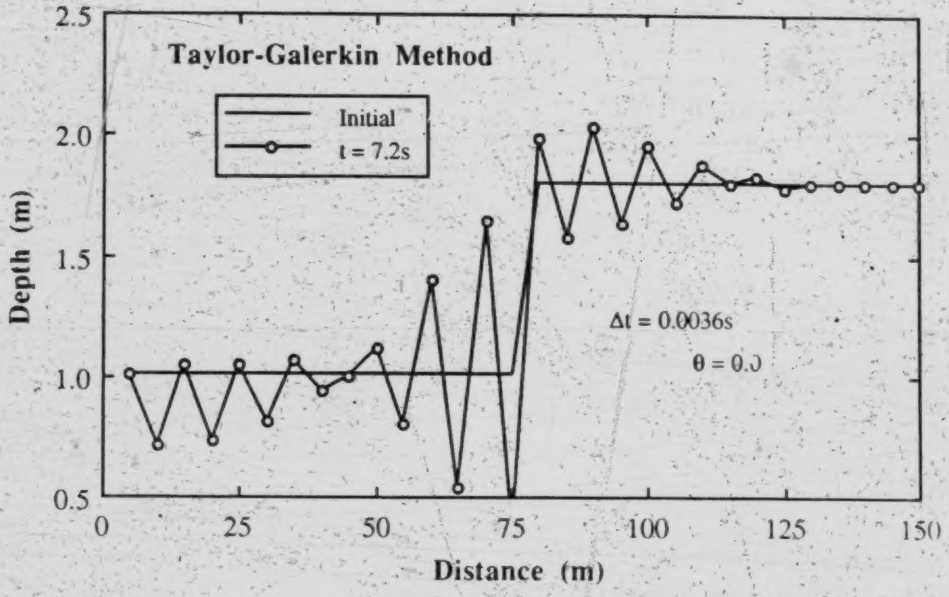


Figure 5.6 Hydraulic jump (t = 0.72s), Taylor-Galerkin method, θ = 0.0, Δt = 0.0036s.



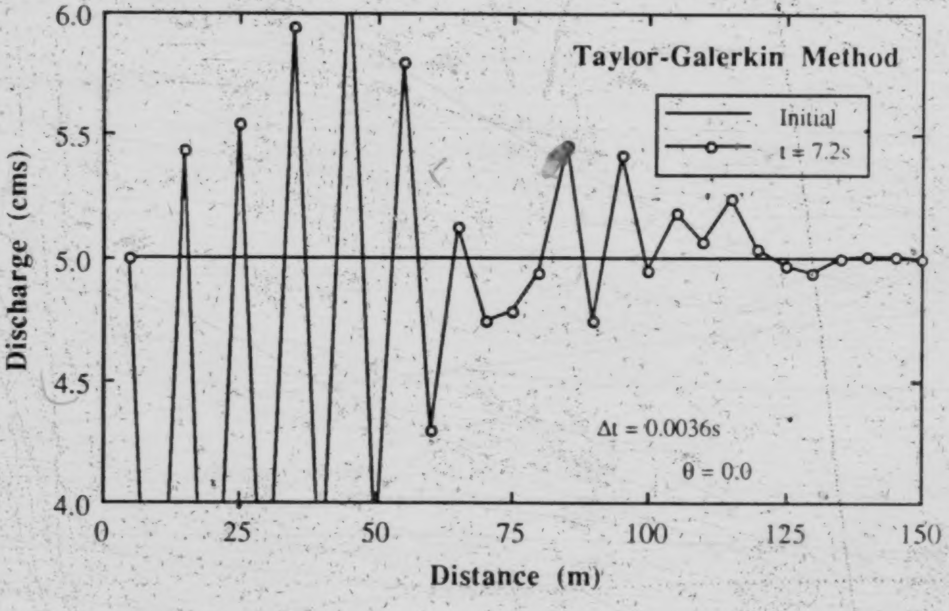


Figure 5.7 Hydraulic jump (t = 7.2s), Taylor-Galerkin method, θ = 0.0, Δt = 0.0036s.

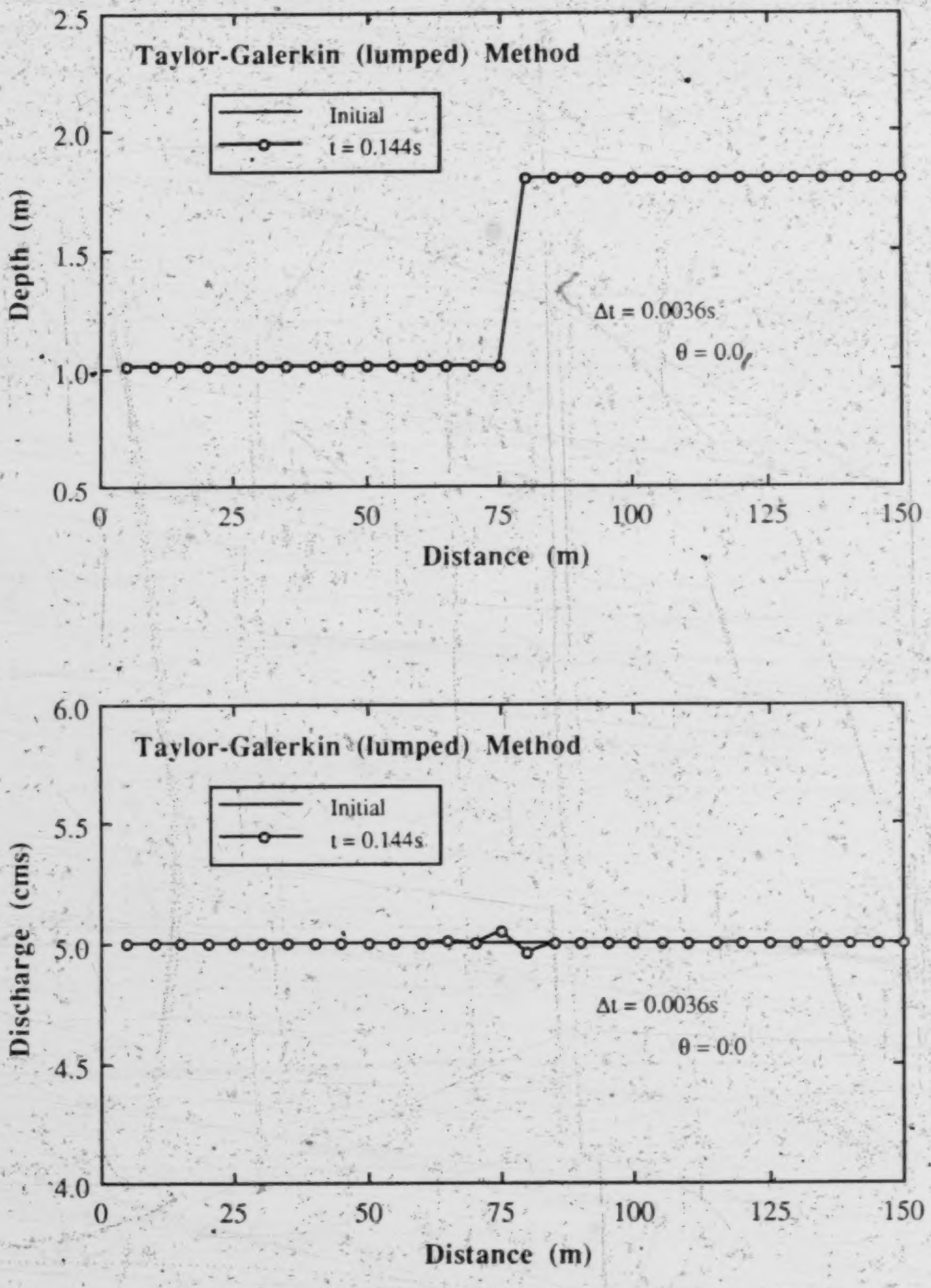


Figure 5.8 Hydraulic jump (t = 0.144s), Taylor-Galerkin (lumped) method, θ = 0.0, Δt = 0.0036s.

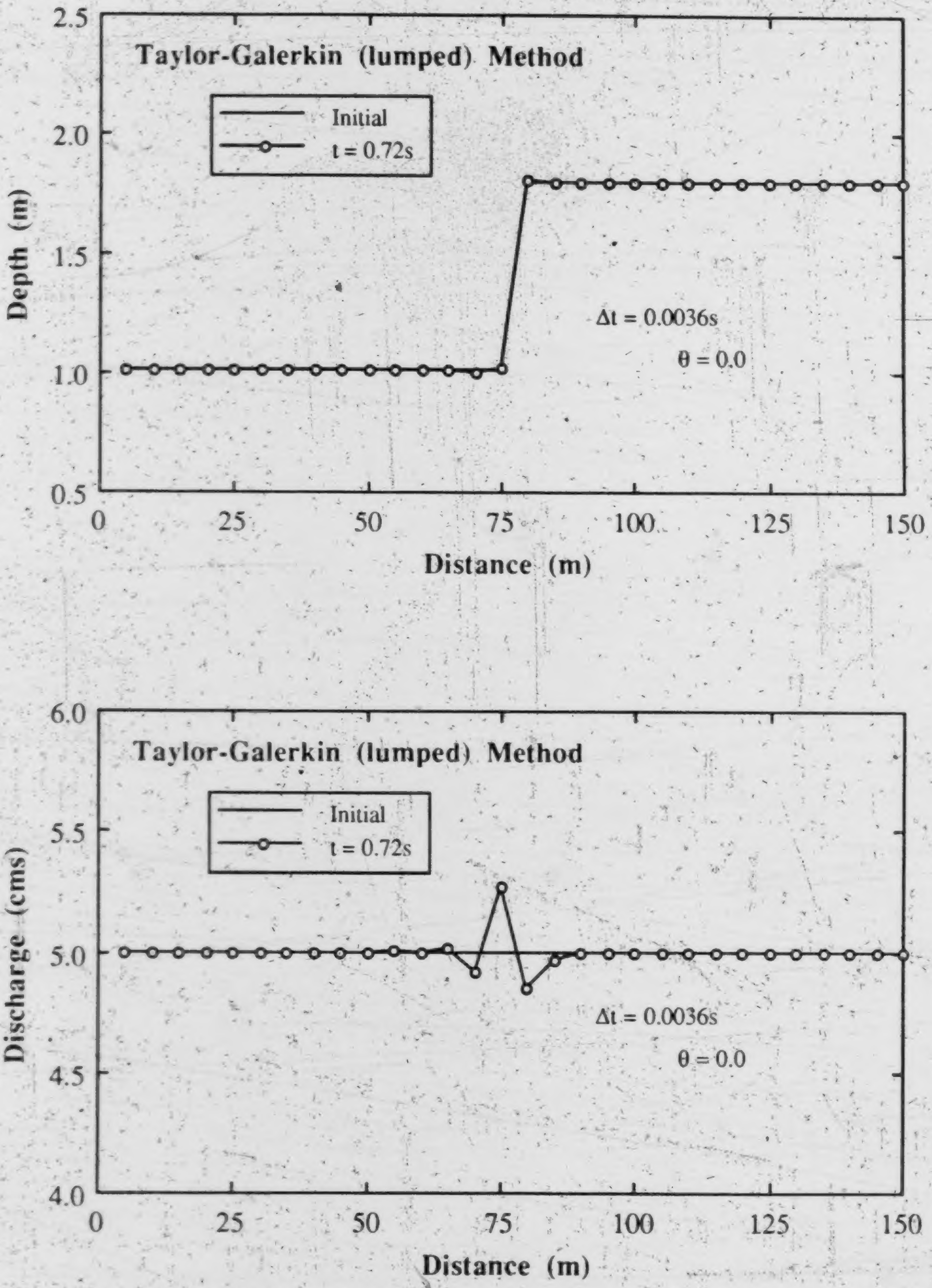
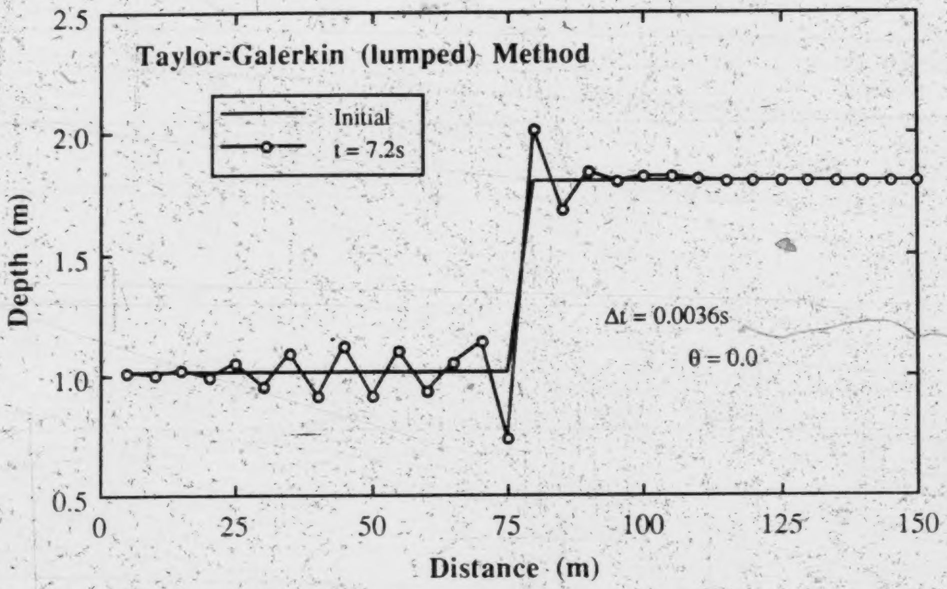


Figure 5.9 Hydraulic jump (t = 0.72s), Taylor-Galerkin (lumped) method, $\theta = 0.0$, $\Delta t = 0.0036$s.



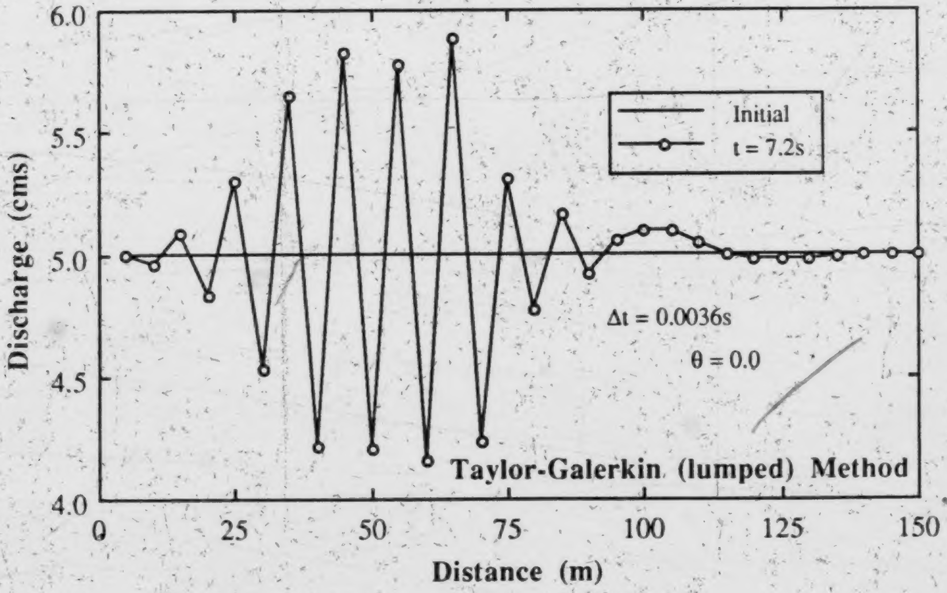
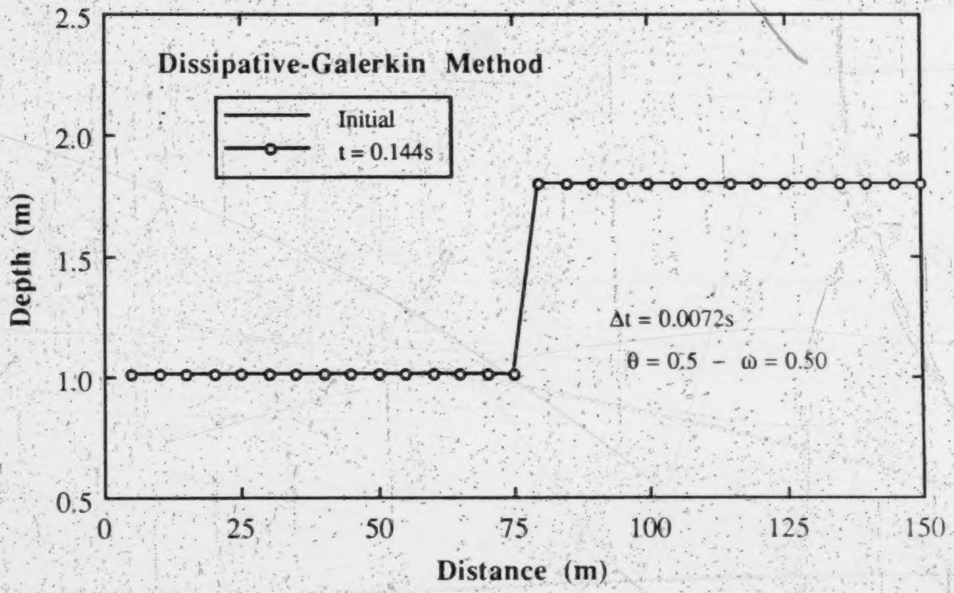


Figure 5.10 Hydraulic jump (t = 7.2s), Taylor-Galerkin (lumped) method, θ = 0.0, Δt = 0.0036s.



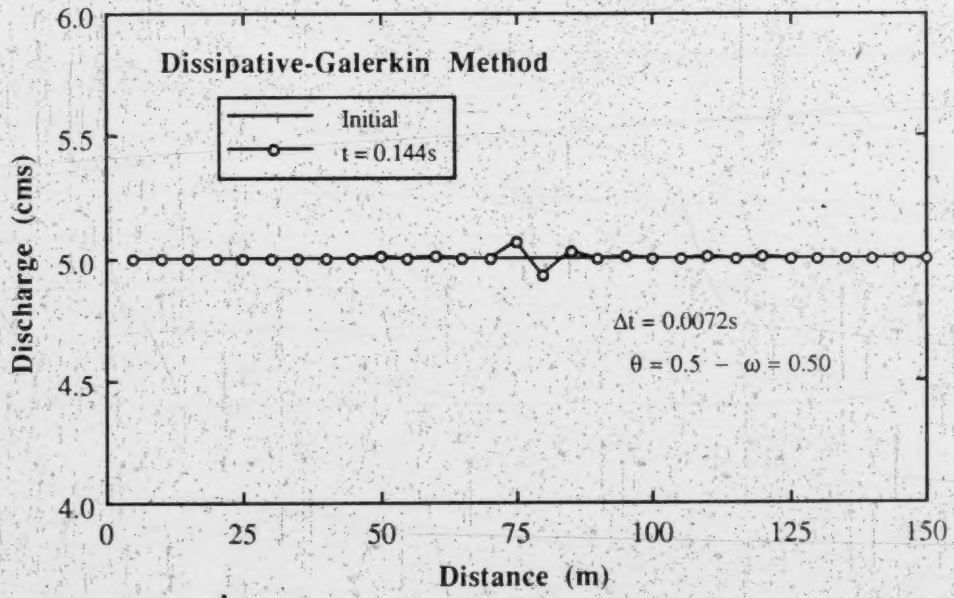
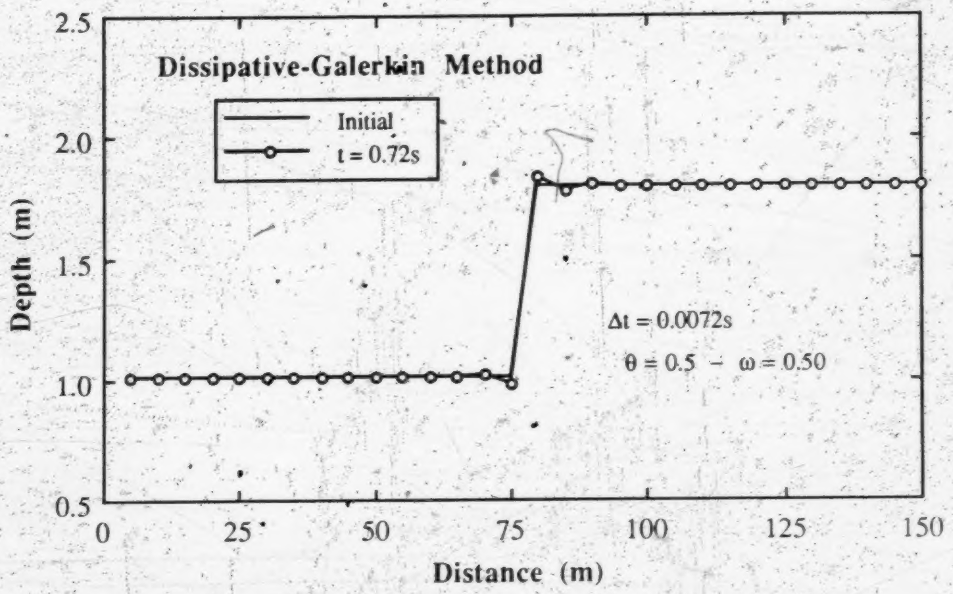


Figure 5.11 Hydraulic jump (t = 0.144s), Dissipative-Galerkin method, $\theta = 0.5$, $\Delta t = 0.0072s$, $\omega = 0.50$.



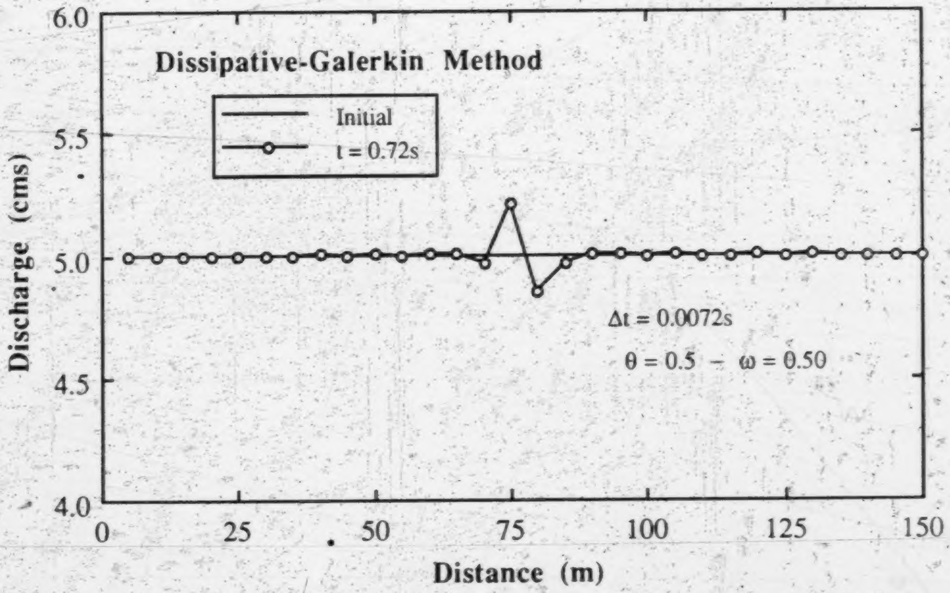


Figure 5.12 Hydraulic jump (t = 0.72s), Dissipative-Galerkin method, $\theta = 0.5$, $\Delta t = 0.0072$s, $\omega = 0.50$.

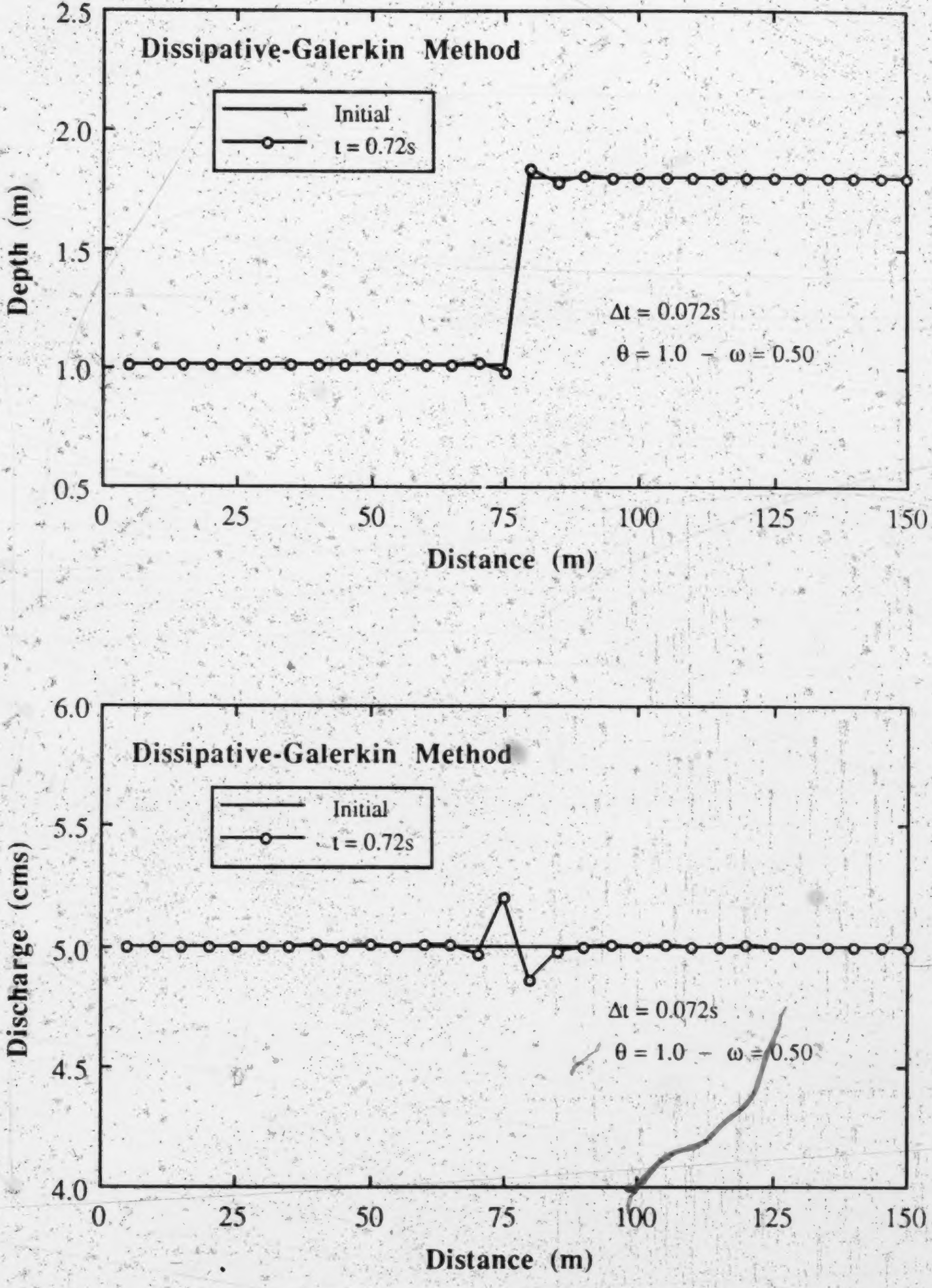
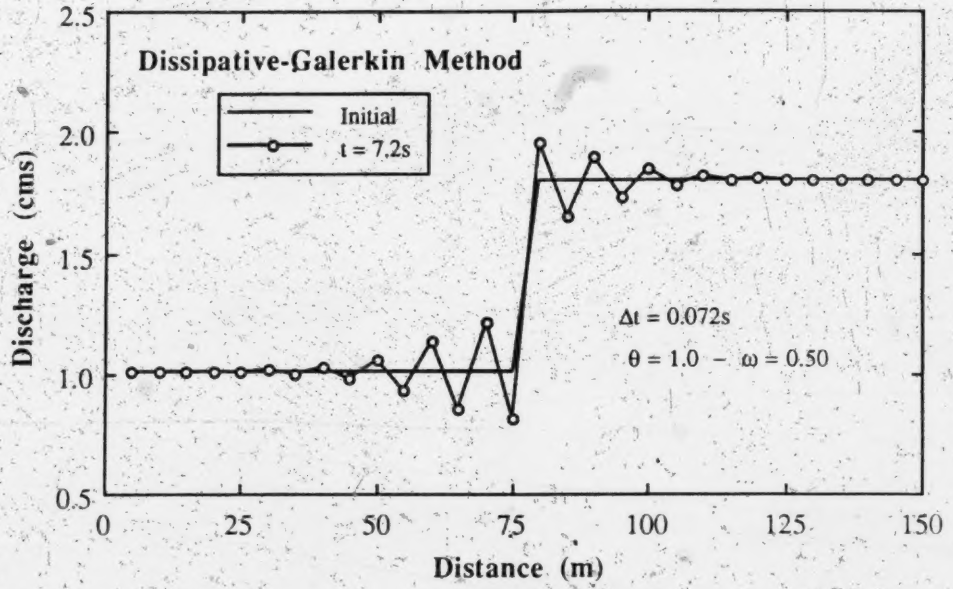


Figure 5.13 Hydraulic jump (t = 0.72s), Dissipative-Galerkin method, θ = 1.0, Δt = 0.072s, ω = 0.50.



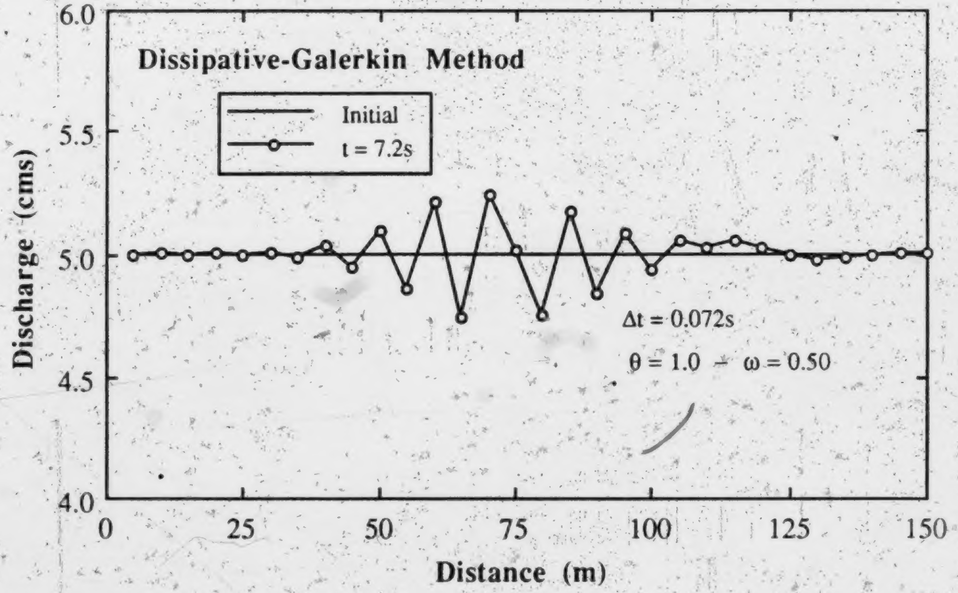
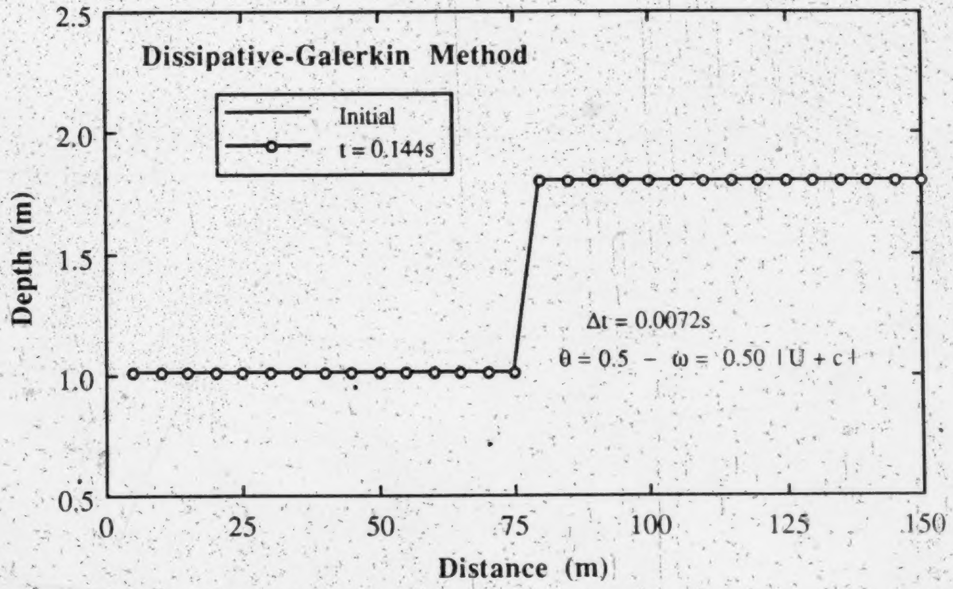


Figure 5.14 Hydraulic jump (t = 7.2s), Dissipative-Galerkin method, θ = 1.0, Δt = 0.072s, ω = 0.50.



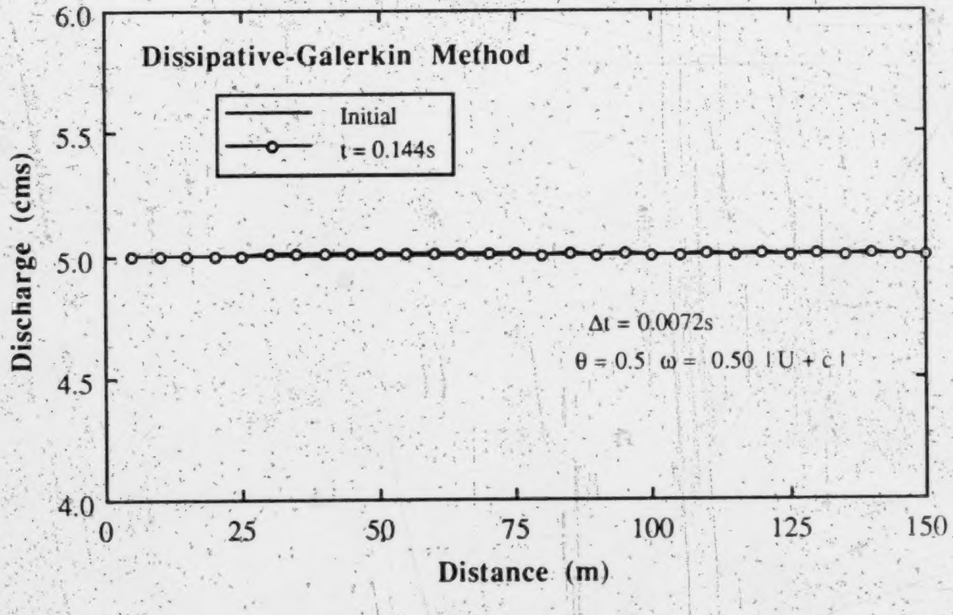
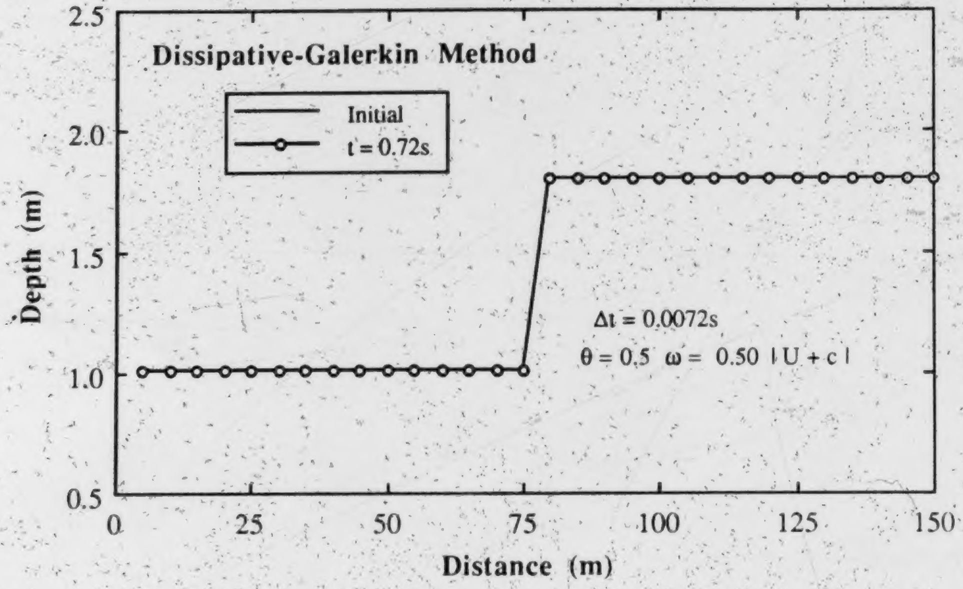


Figure 5.15 Hydraulic jump ($t = 0.144s$), Dissipative-Galerkin method, $\theta = 0.5$, $\Delta t = 0.0072s$, $\omega = 0.50 \mid U+c \mid$.



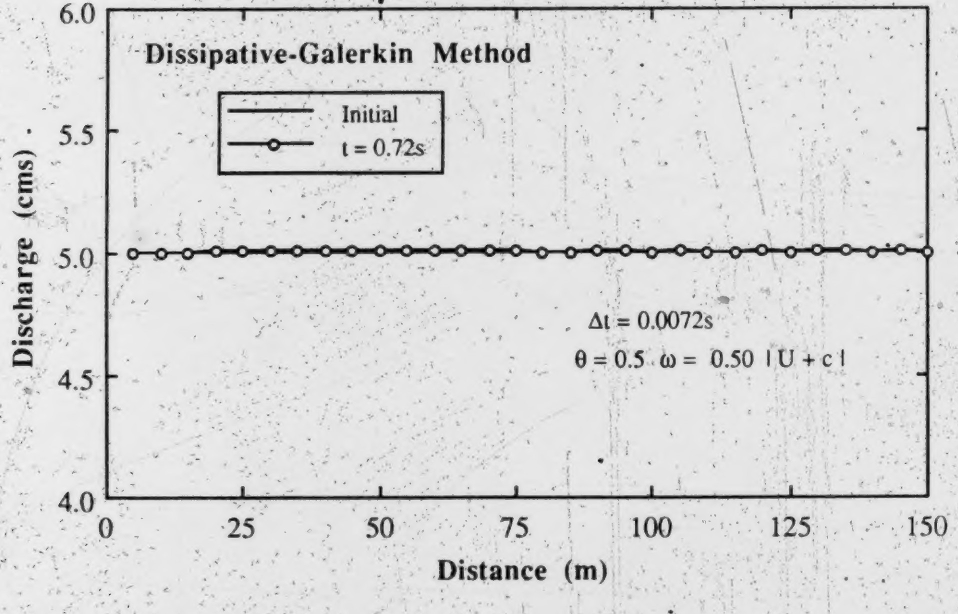
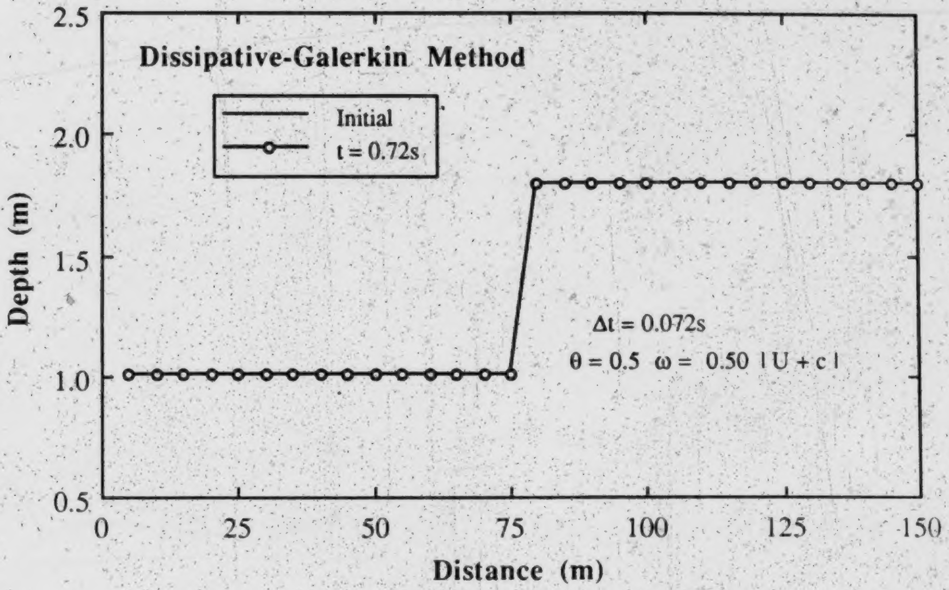


Figure 5.16 Hydraulic jump (t = 0.72s), Dissipative-Galerkin method, θ = 0.5, Δt = 0.0072s, $\bar{\omega}$ = 0.50 | U+c |.



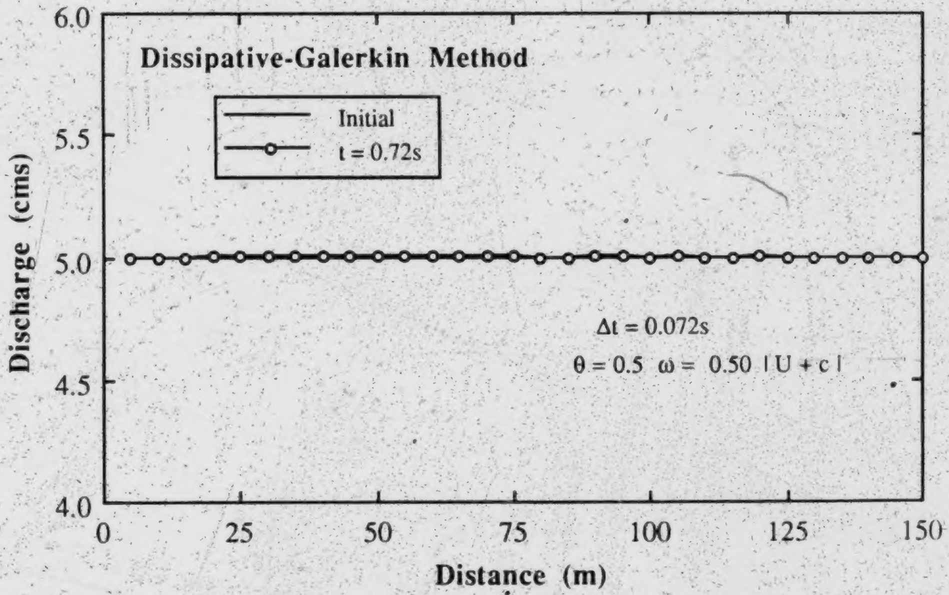
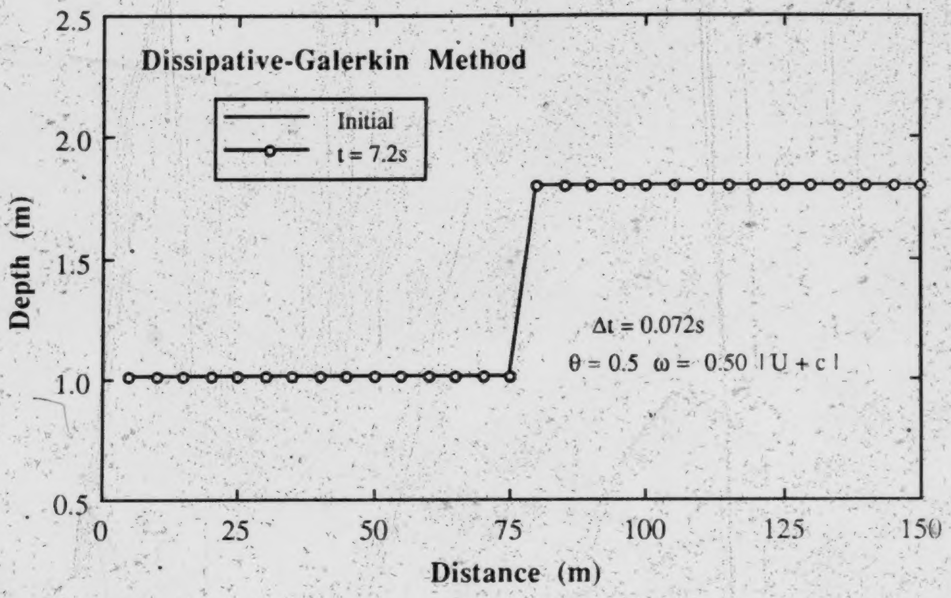


Figure 5.17 Hydraulic jump ($t = 0.72s$), Dissipative-Galerkin method, $\theta = 1.0$, $\Delta t = 0.072s$, $\omega = 0.50 \mid U+c \mid$.



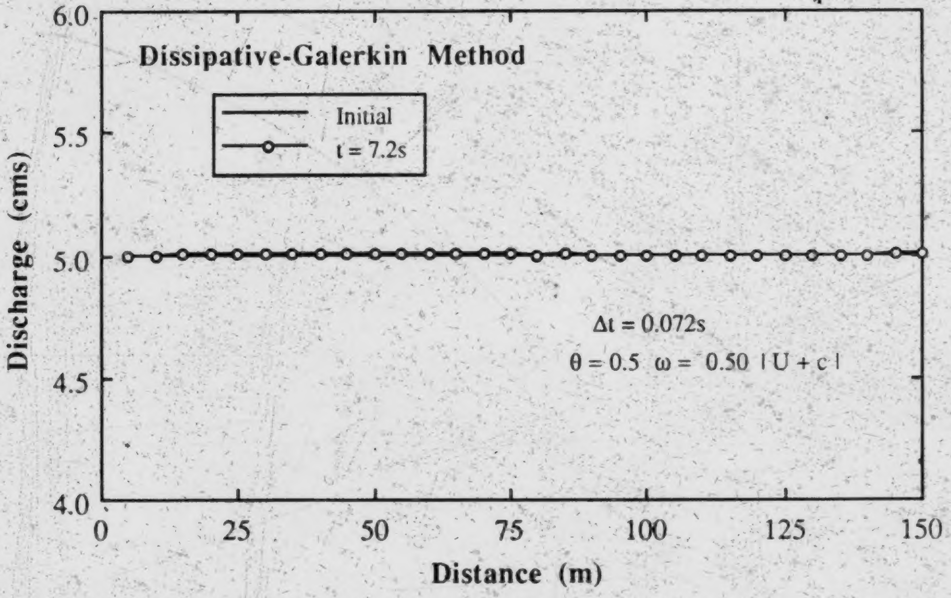
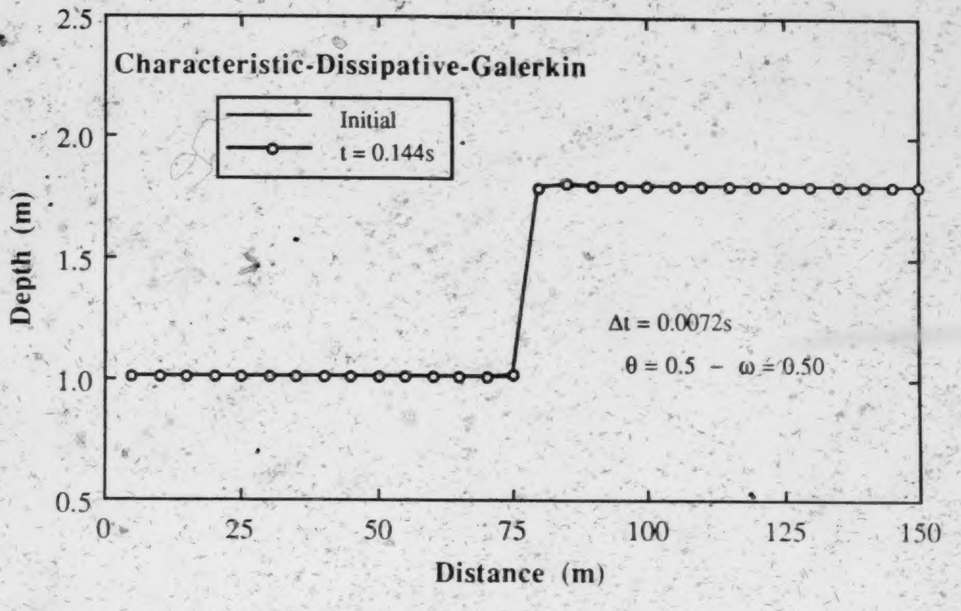


Figure 5.18 Hydraulic jump (t = 7.2s), Dissipative-Galerkin method, θ = 1.0, Δt = 0.072s, ω = 0.50 | U+c |.



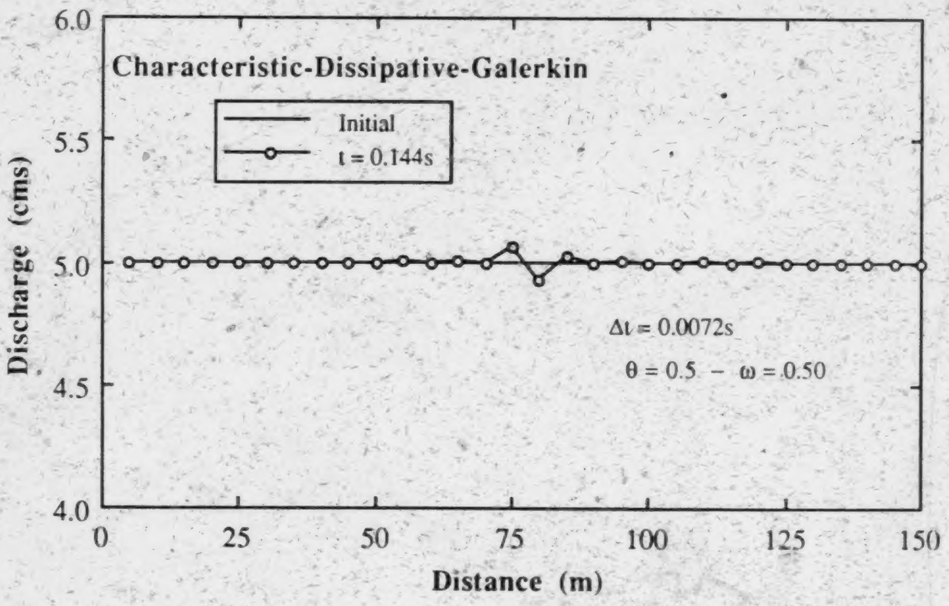


Figure 5.19 Hydraulic jump ($t = 0.144s$), Characteristic-Dissipative-Galerkin method, $\theta = 0.5$, $\Delta t = 0.0072s$, $\omega = 0.50$.

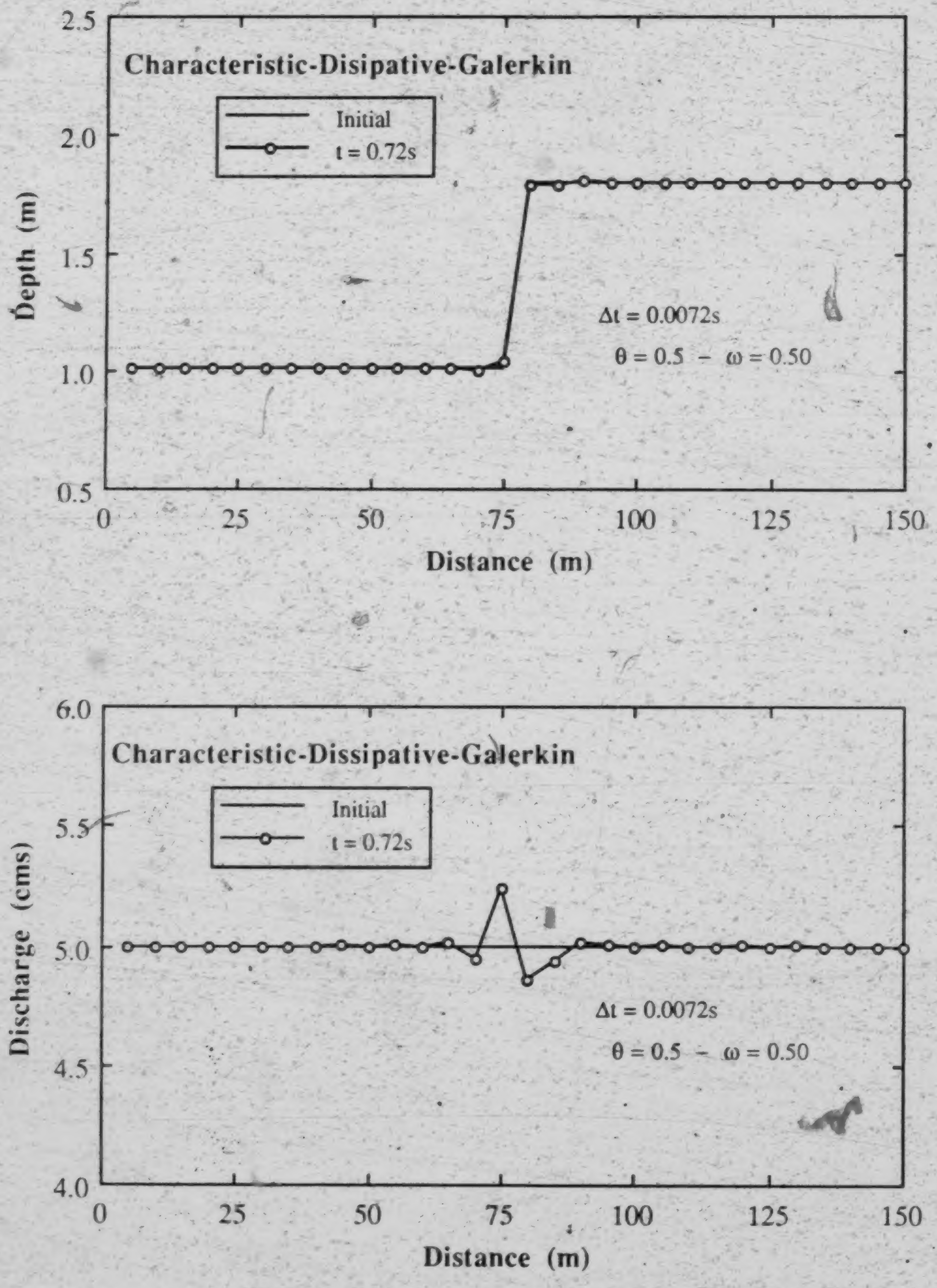
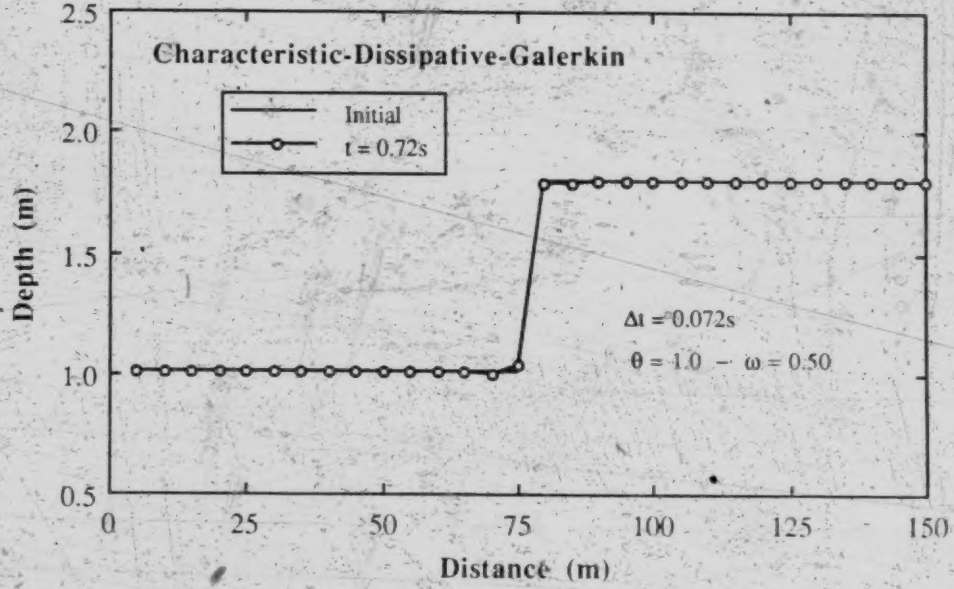


Figure 5.20 Hydraulic jump (t = 0.72s), Characteristic-Dissipative-Galerkin method, $\theta = 0.5$, $\Delta t = 0.0072s$, $\omega = 0.50$.



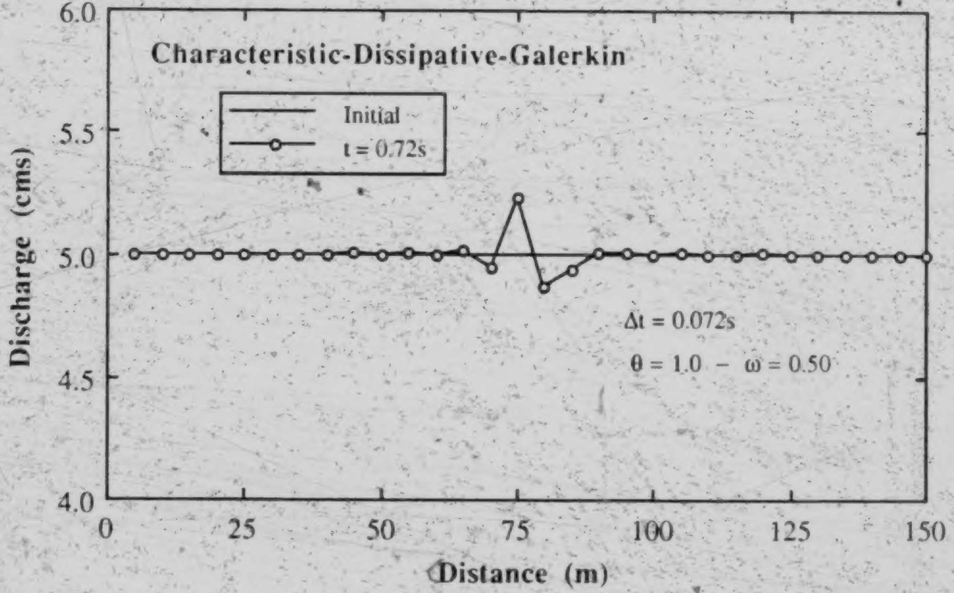
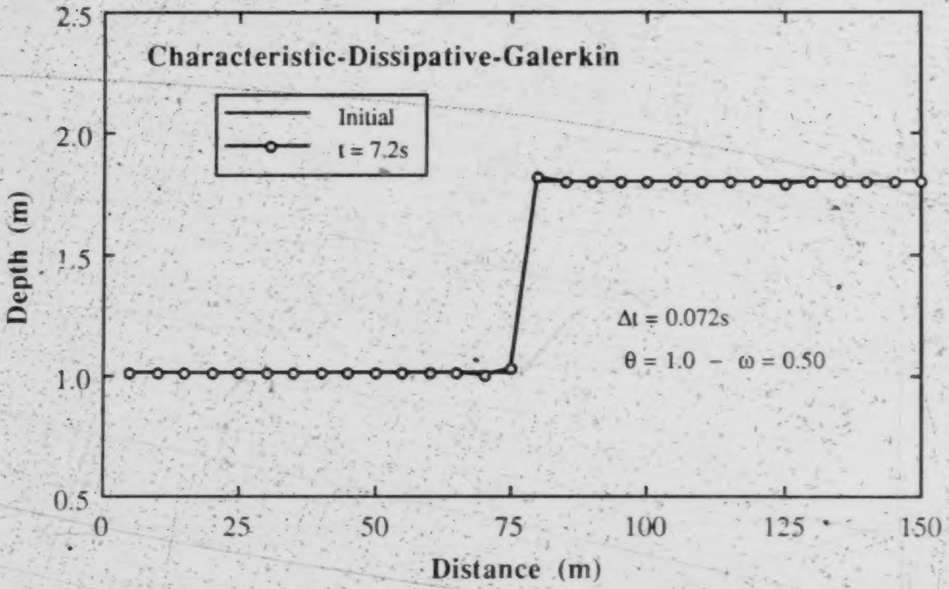


Figure 5.21 Hydraulic jump (t = 0.72s), Characteristic-Dissipative-Galerkin method, $\theta = 1.0$, $\Delta t = 0.072s$, $\omega = 0.50$.



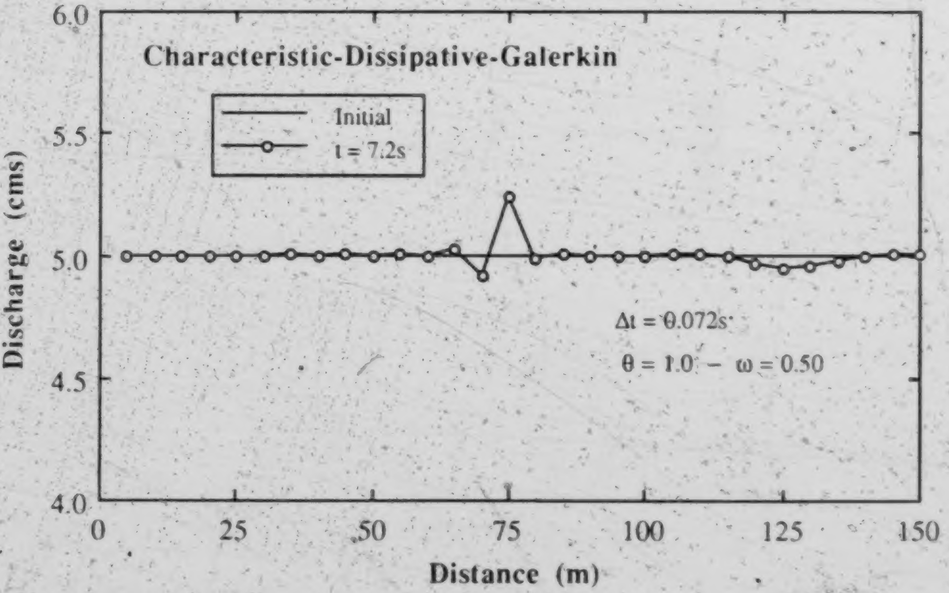


Figure 5.22 Hydraulic jump (t = 7.2s), Characteristic-Dissipative-Galerkin method, θ = 1.0, Δt = 0.072s, ω = 0.50.

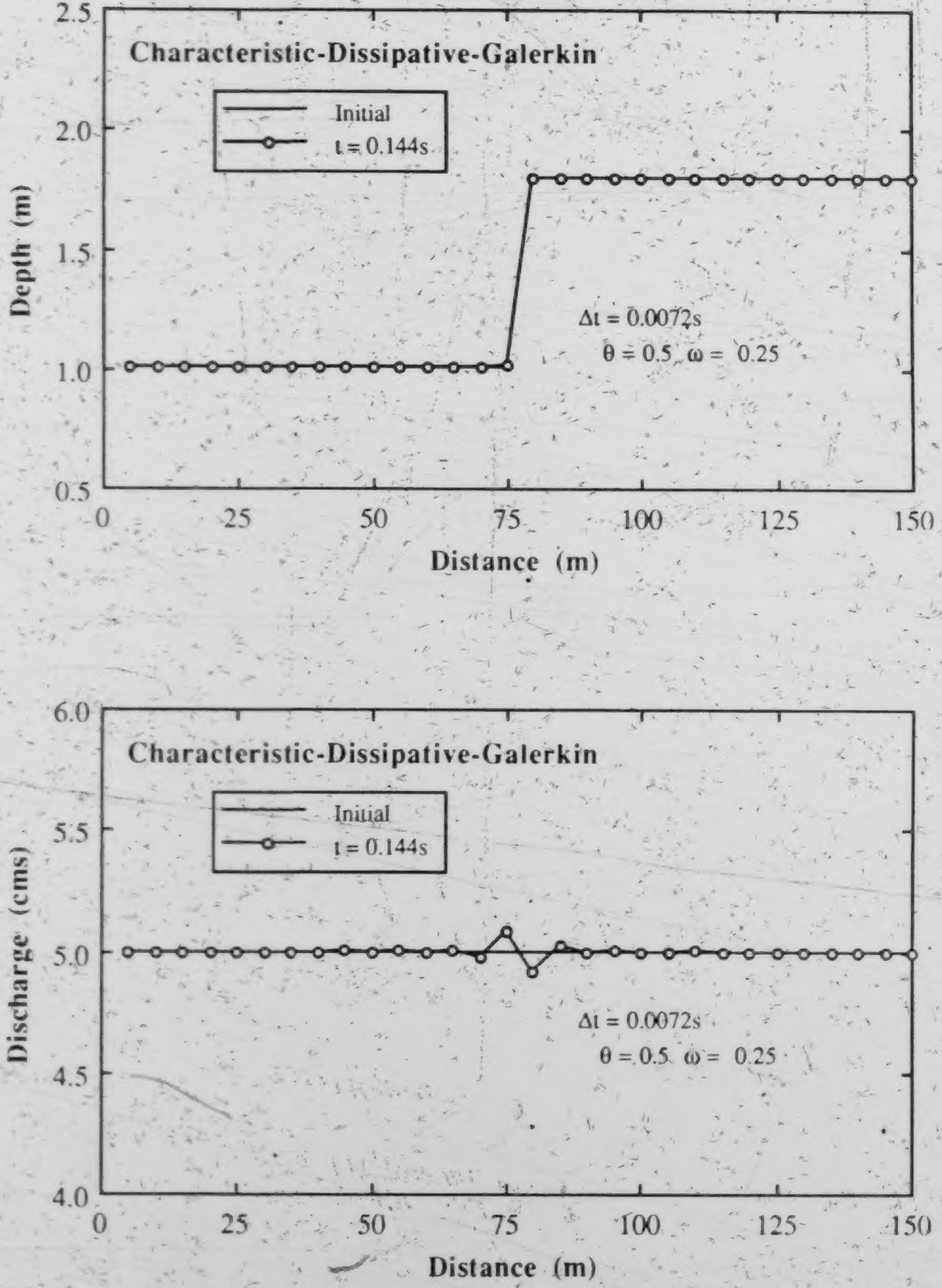
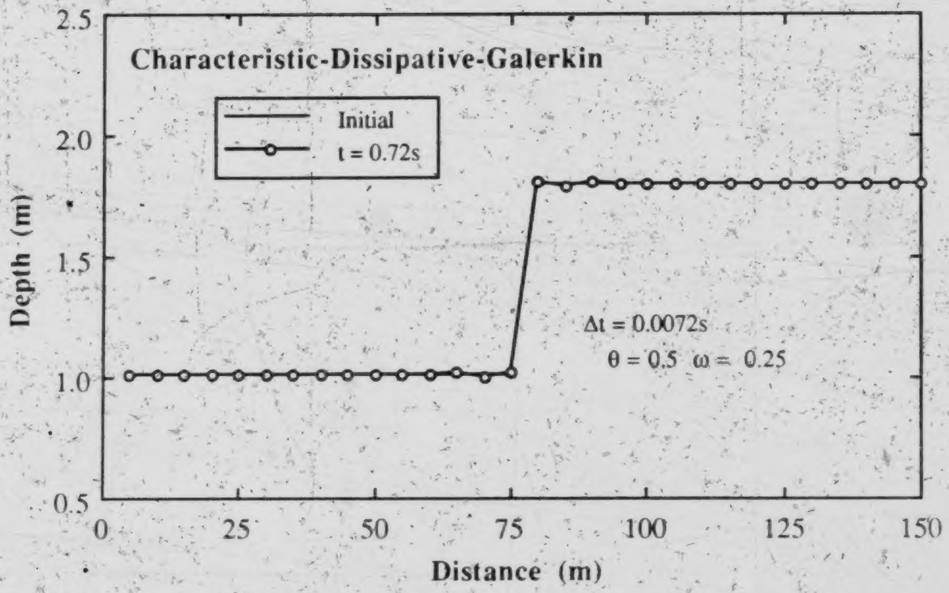


Figure 5.23 Hydraulic jump ($t = 0.144s$), Characteristic-Dissipative-Galerkin method, $\theta = 0.5$, $\Delta t = 0.0072s$, $\omega = 0.25$.



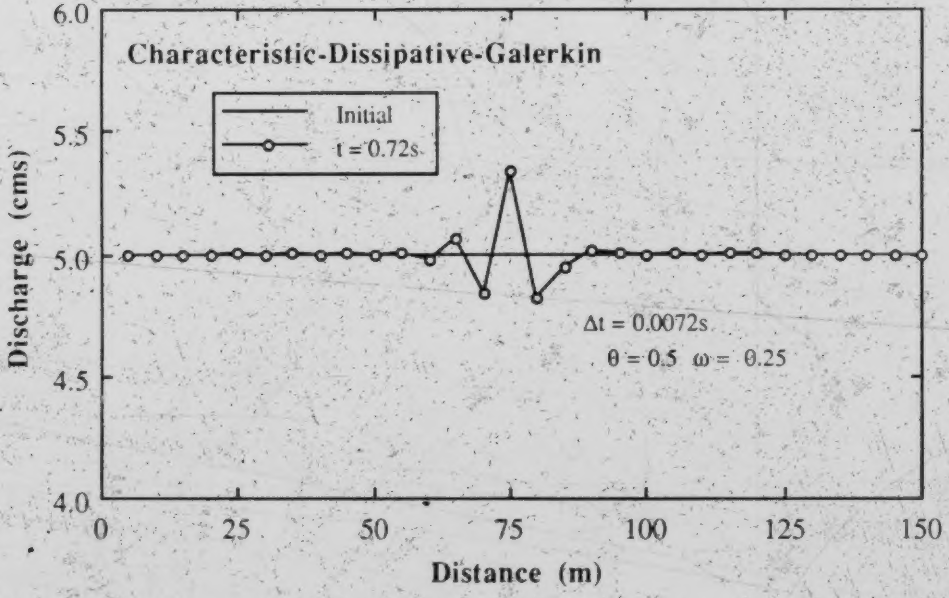


Figure 5.24 Hydraulic jump (t = 0.72s), Characteristic-Dissipative-Galerkin method, θ = 0.5, Δt = 0.0072s, ω = 0.25.

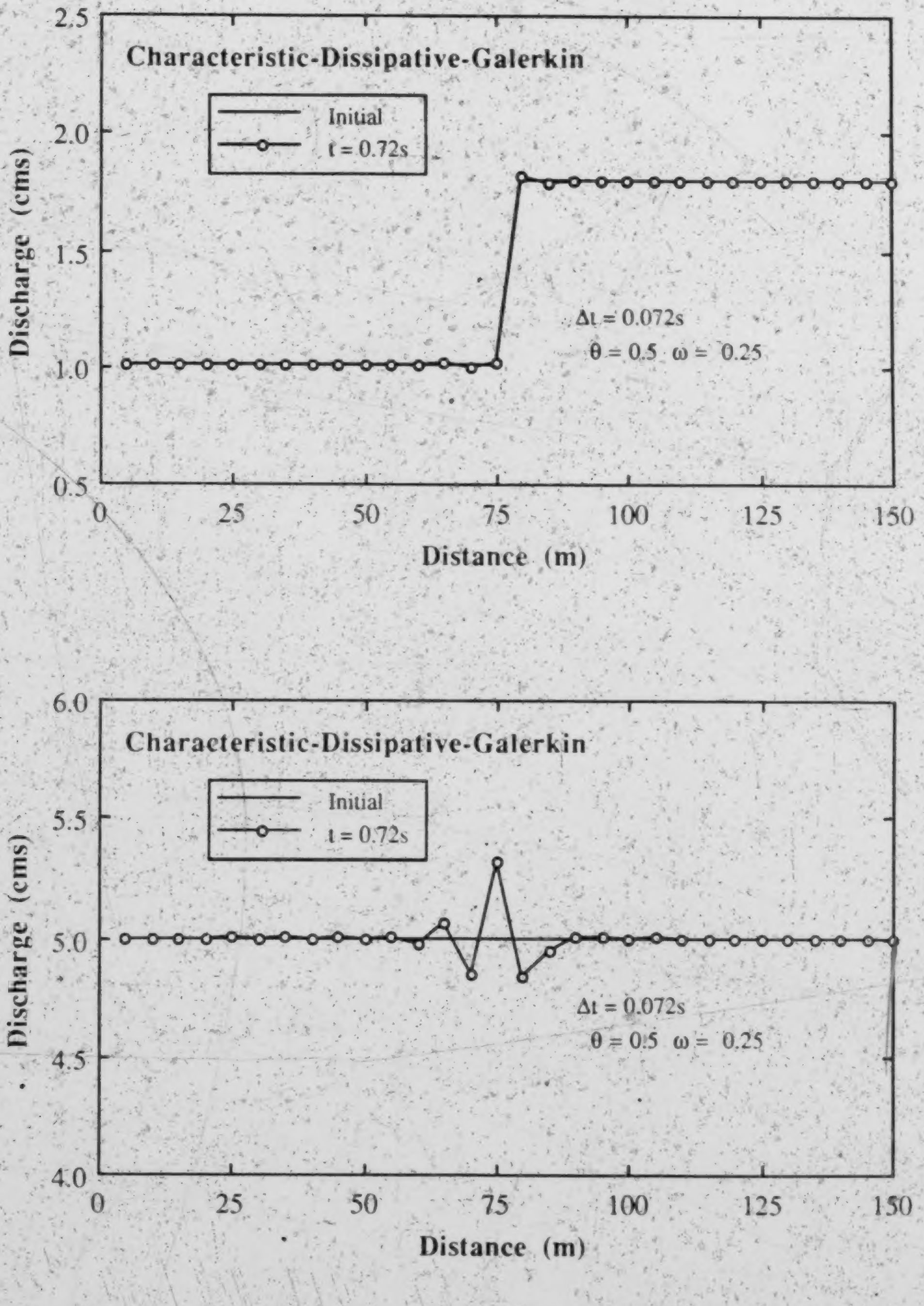
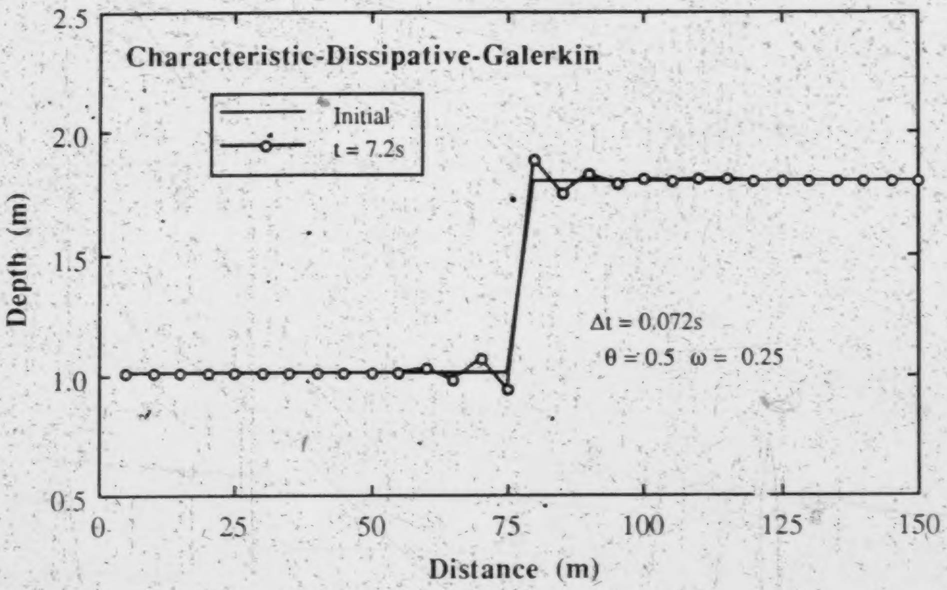


Figure 5.25 Hydraulic jump (t = 0.72s), Characteristic-Dissipative-Galerkin method, θ = 1.0, Δt = 0.072s, ω = 0.25



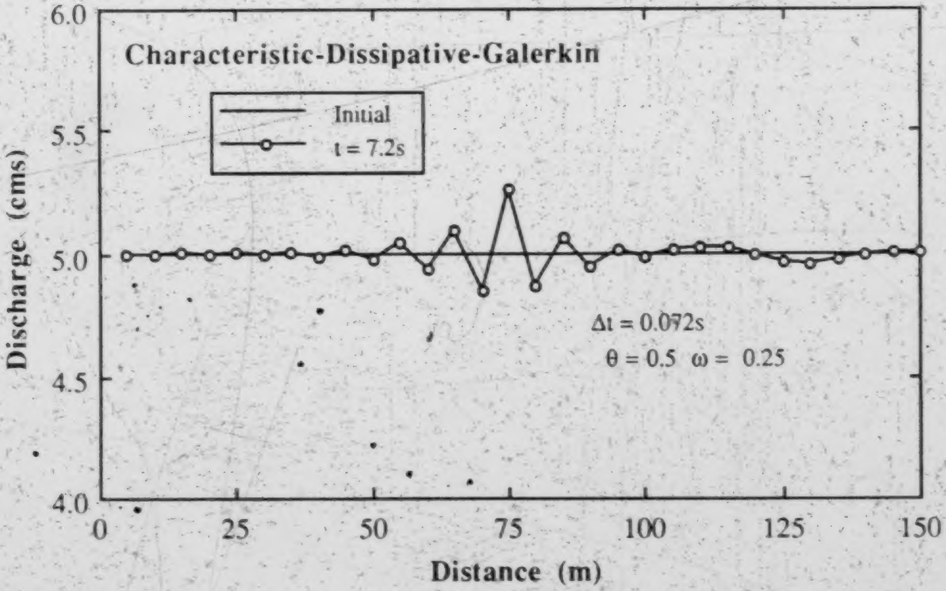
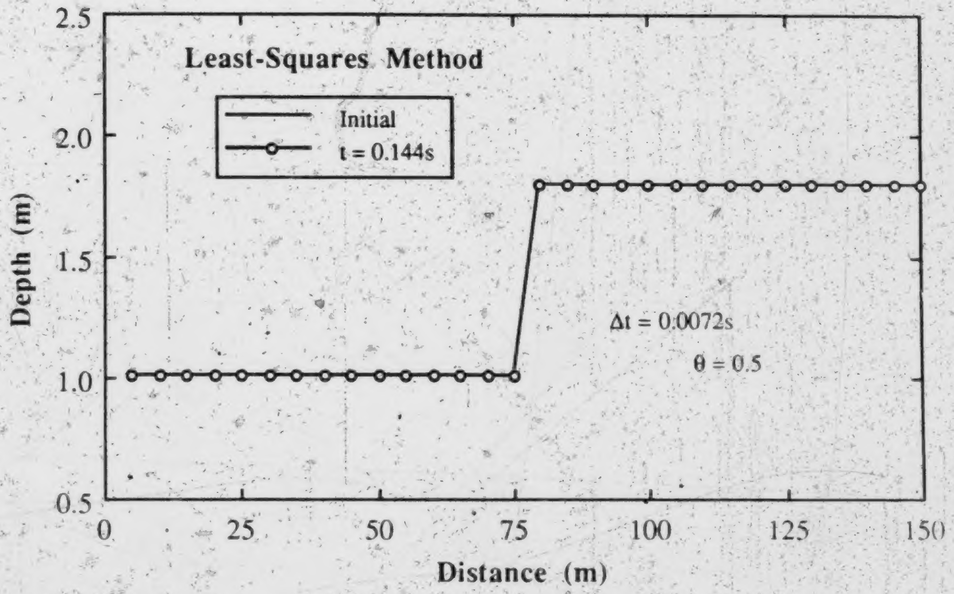


Figure 5.26 Hydraulic jump (t = 7.2s), Characteristic-Dissipative-Galerkin method, θ = 1.0, Δt = 0.072s, ω = 0.25.



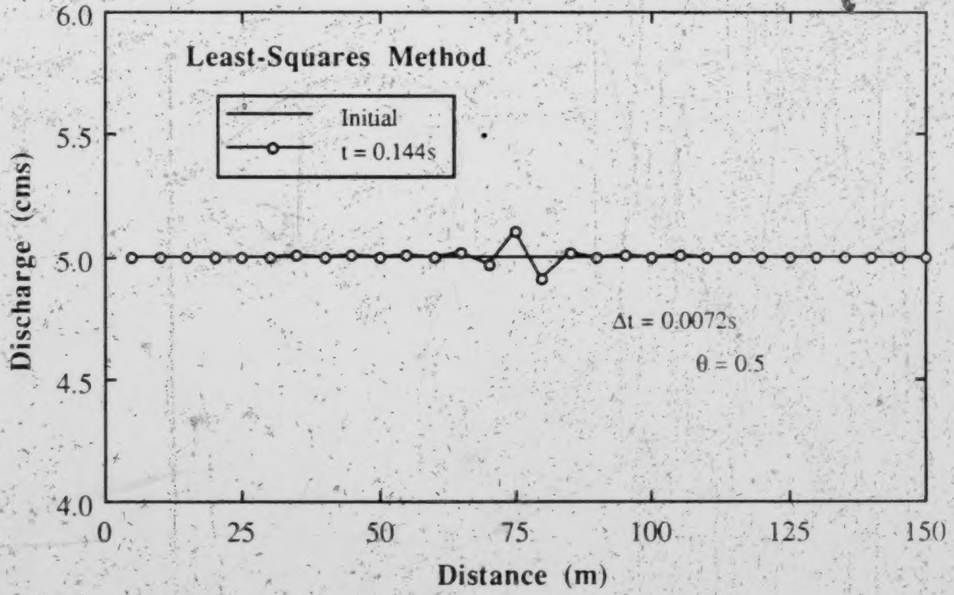
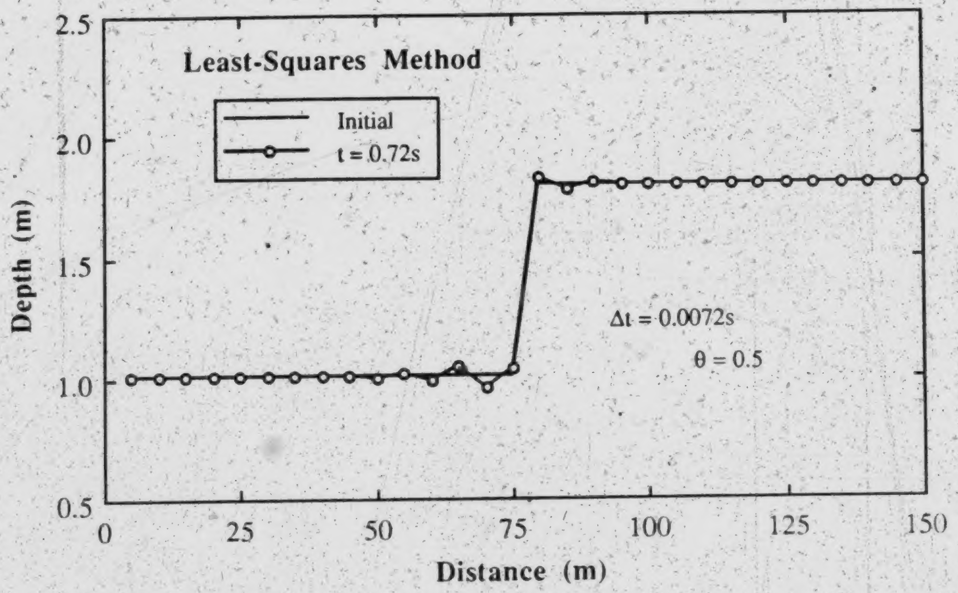


Figure 5.27 Hydraulic jump (t = 0.144s), Least-Squares method, θ = 0.5, Δt = 0.0072s.



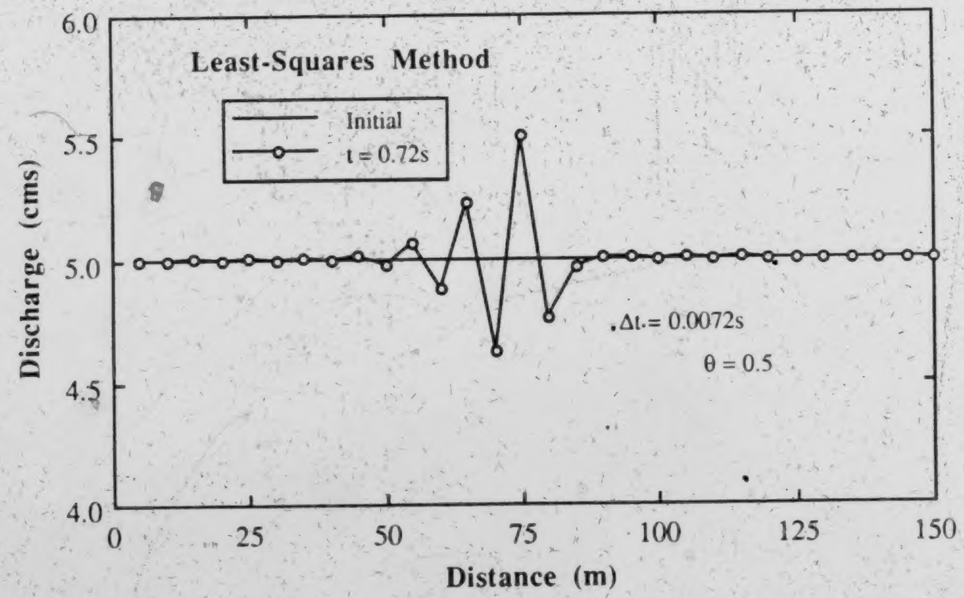
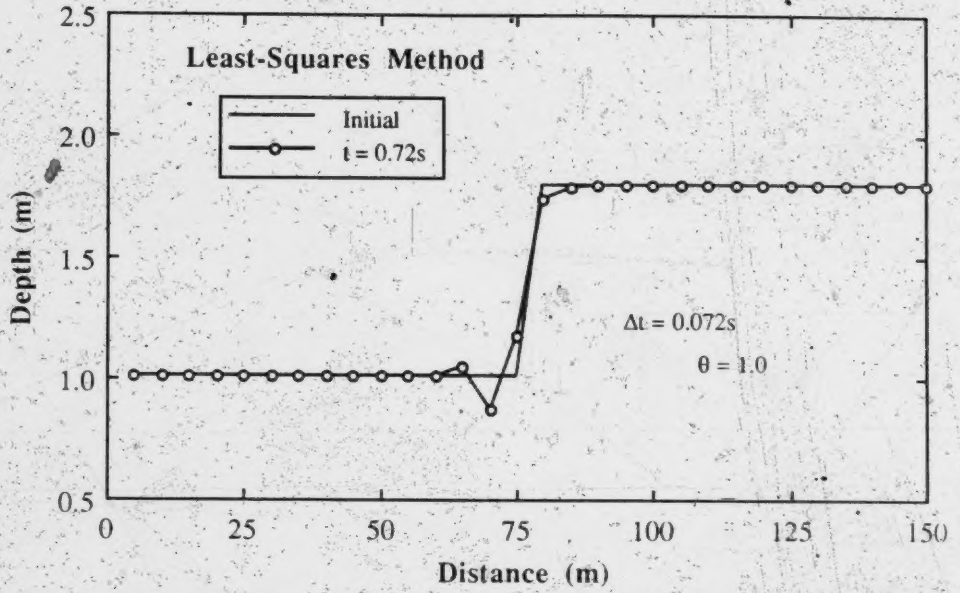


Figure 5.28 Hydraulic jump (t = 0.72s), Least-Squares method, θ = 0.5, Δt = 0.0072s.



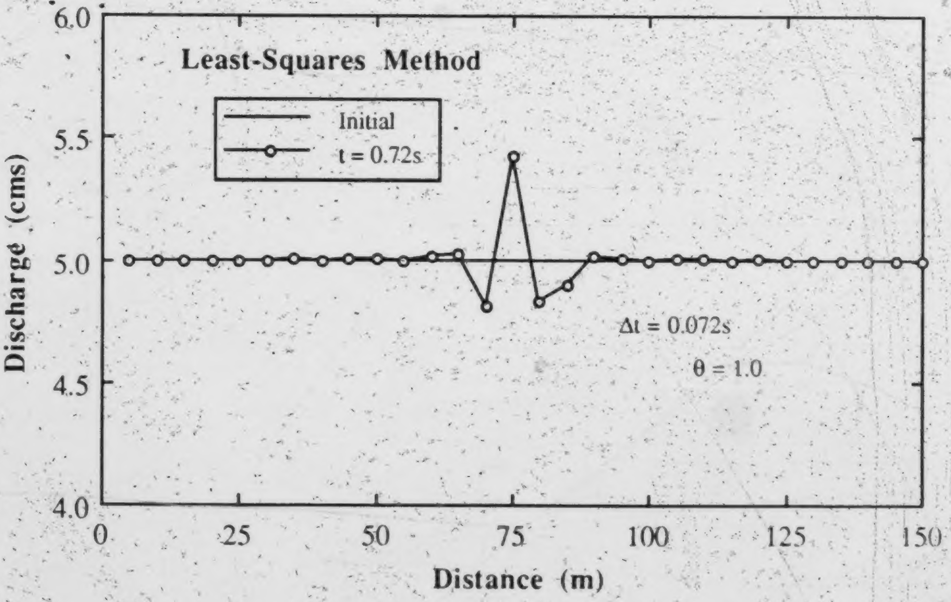
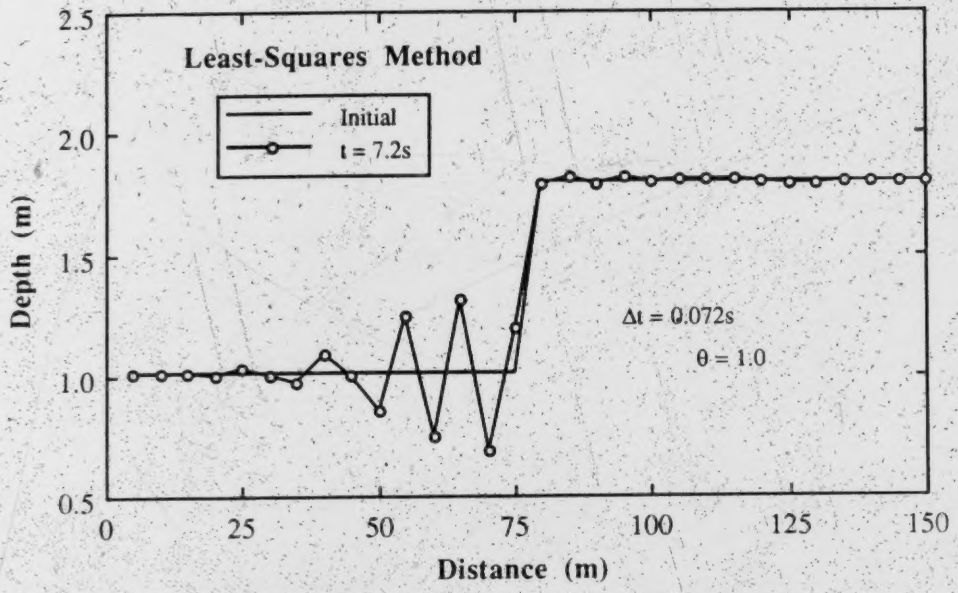


Figure 5.29 Hydraulic jump (t = 0.72s), Least-Squares method, θ = 1.0, Δt = 0.072s.



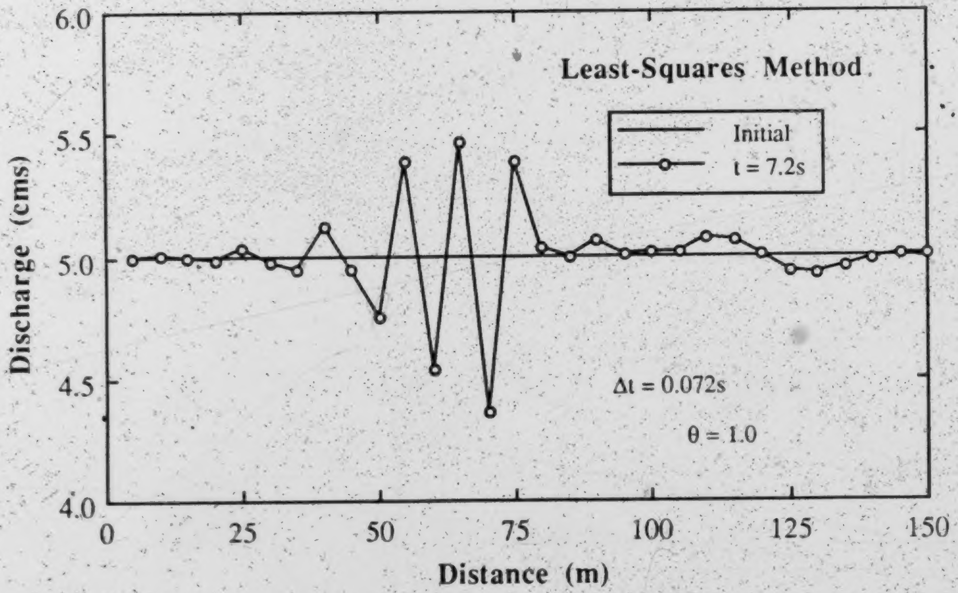
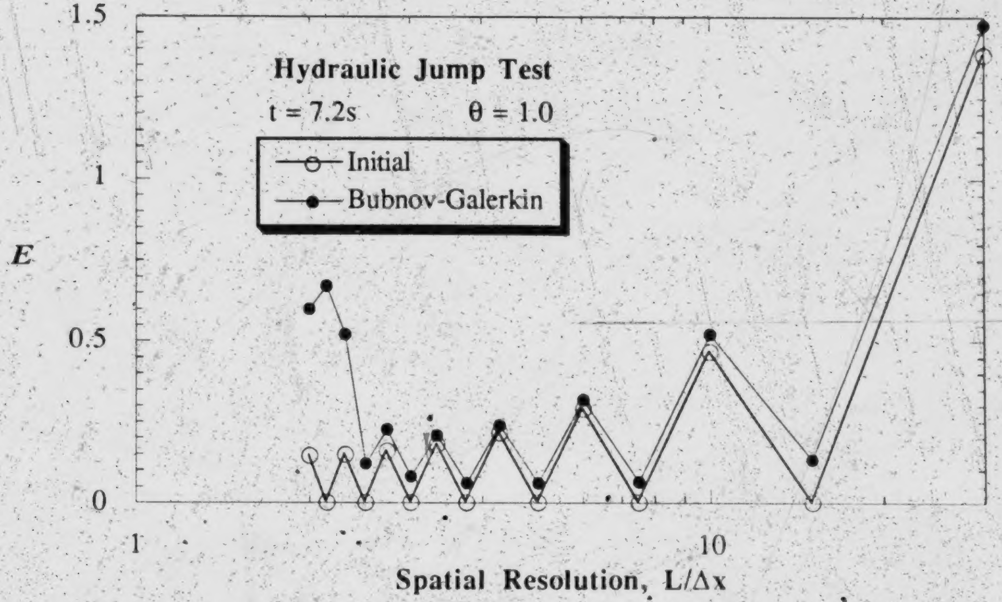
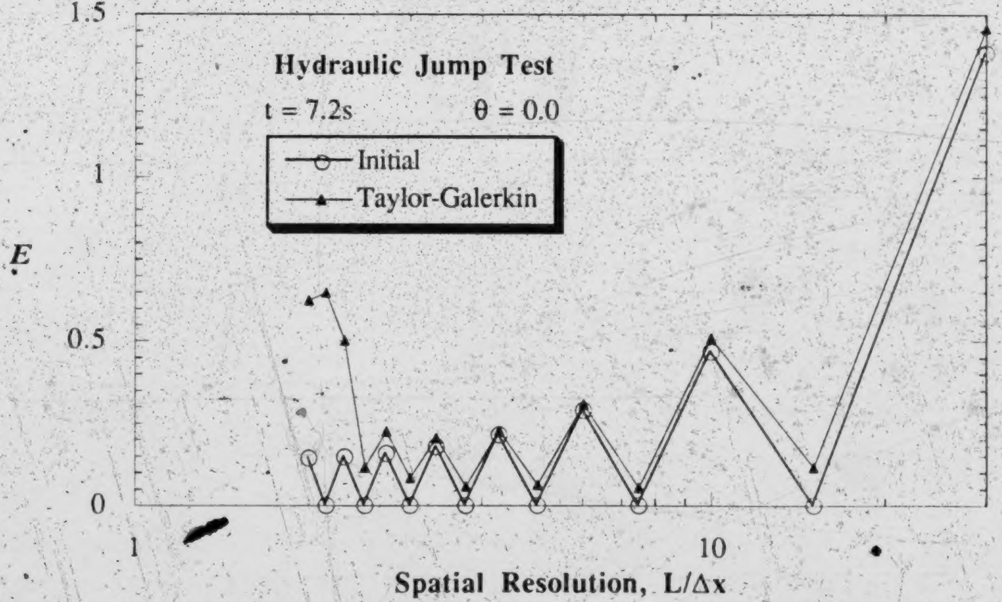


Figure 5.30 Hydraulic jump (t = 7.2s), Least-Squares method, θ = 1.0, Δt = 0.072s.

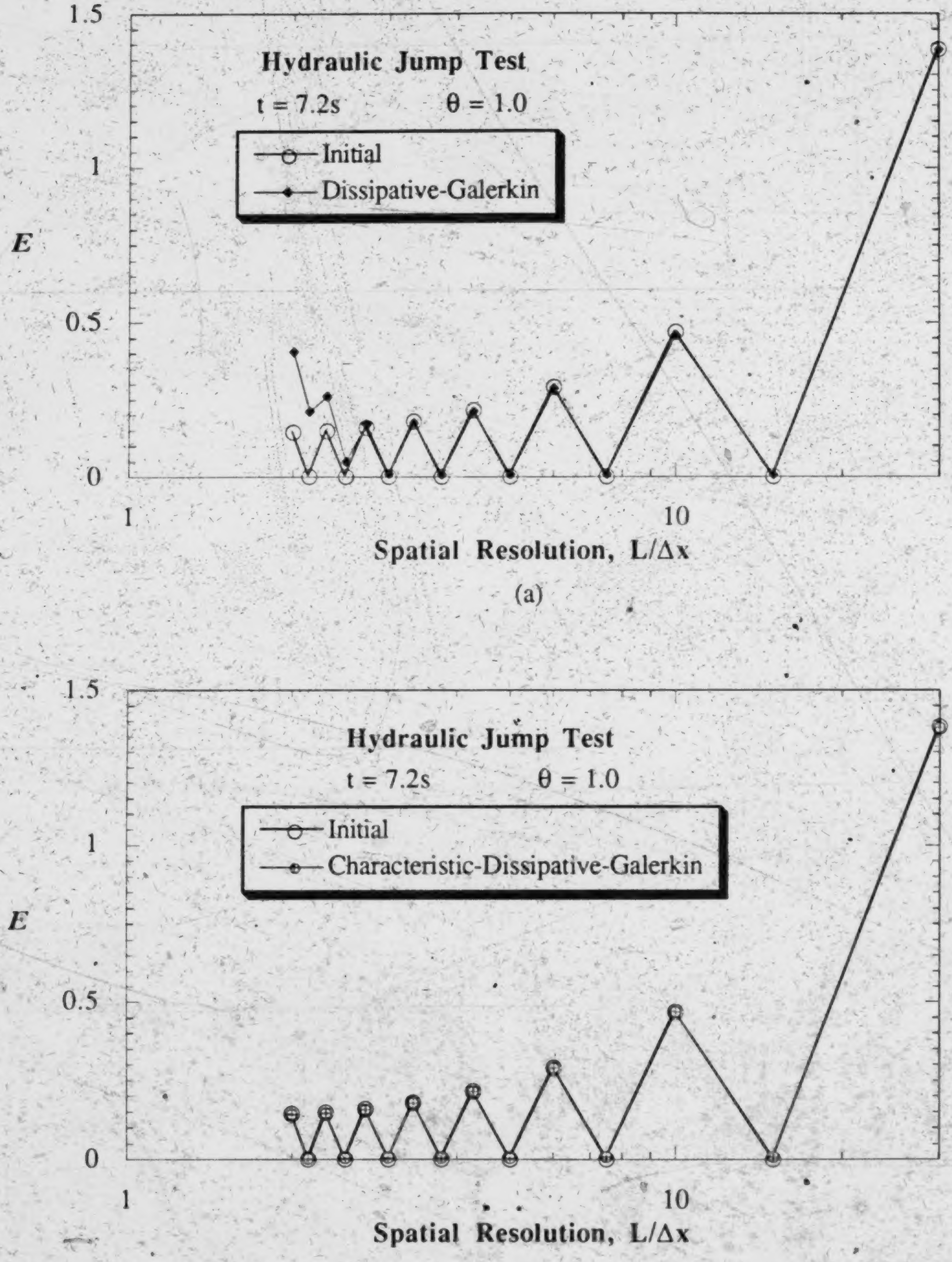


(a)



(b)

Figure 5.31 Results of spectral analysis on hydraulic jump depths (t = 7.2s), (a) Bubnov-Galerkin method, θ = 1.0, Δt = 0.072s; (b) Taylor-Galerkin method, θ = 0.0, Δt = 0.0036s.



(a)

(b)

Figure 5.32 Results of spectral analysis on hydraulic jump depths (t = 7.2s), θ = 1.0, Δt = 0.072s, ω = 0.50, (a) Dissipative-Galerkin method; (b) Characteristic-Dissipative-Galerkin method.

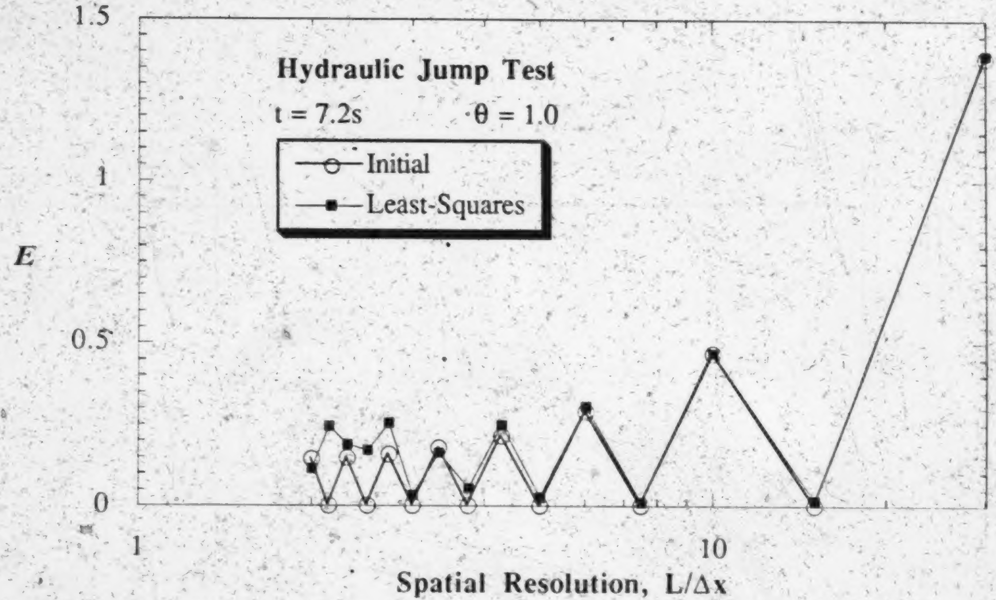


Figure 5.33 Hydraulic jump (t = 7.2s), Least-Squares method, θ = 1.0, Δt = 0.072s.

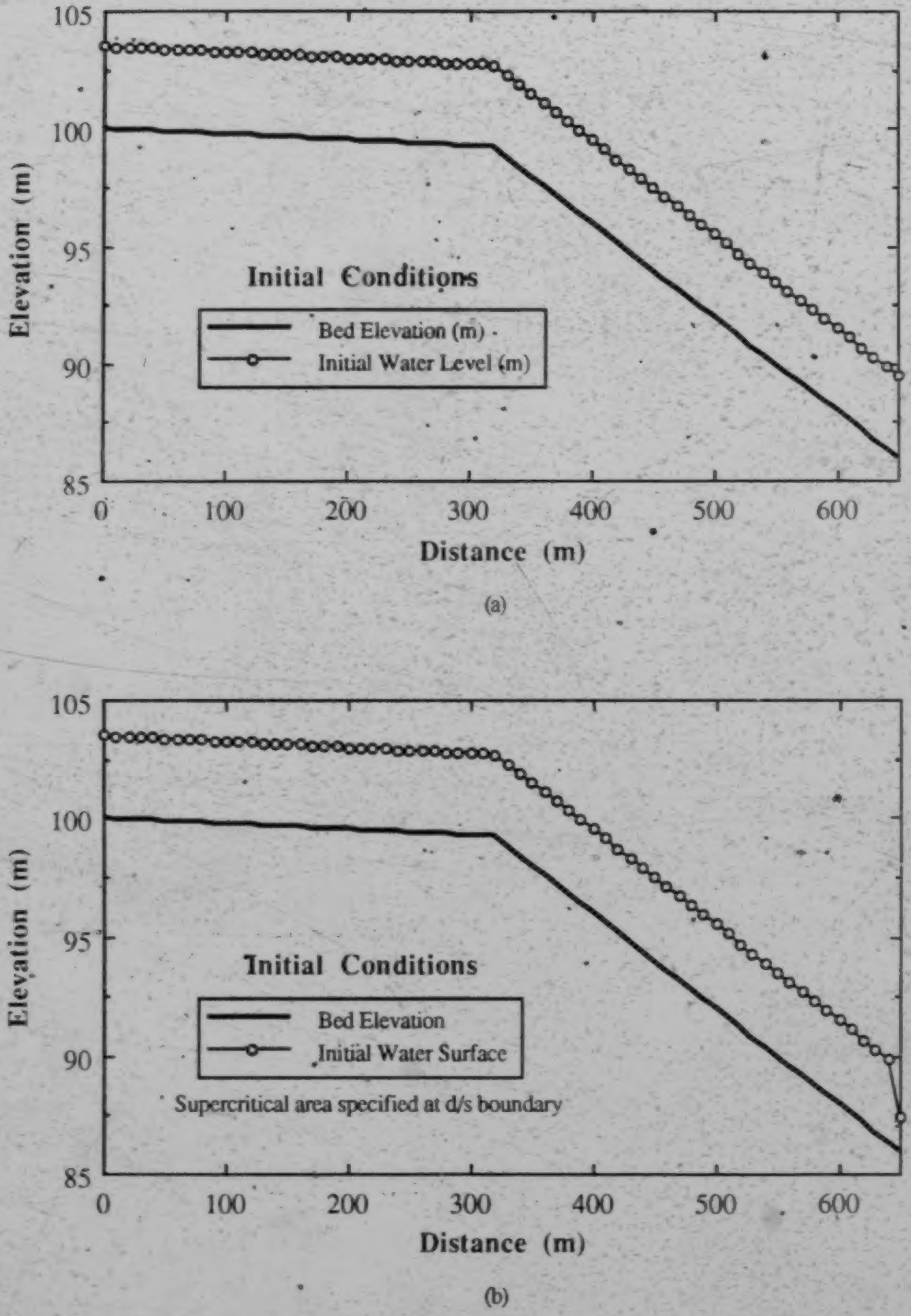
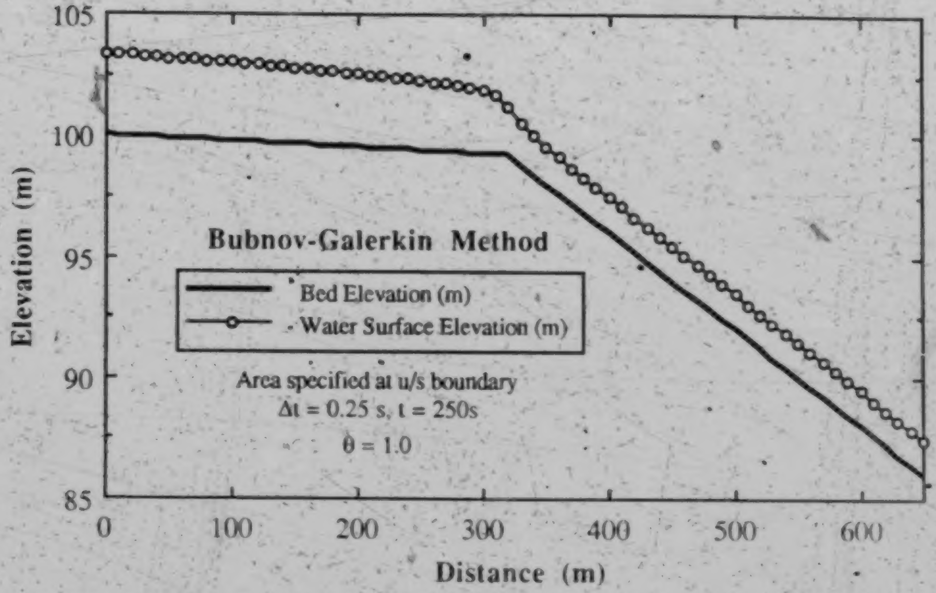


Figure 5.34 Transition from a mild to a steep channel , initial conditions: (a) implicit schemes; (b) Taylor-Galerkin tests.



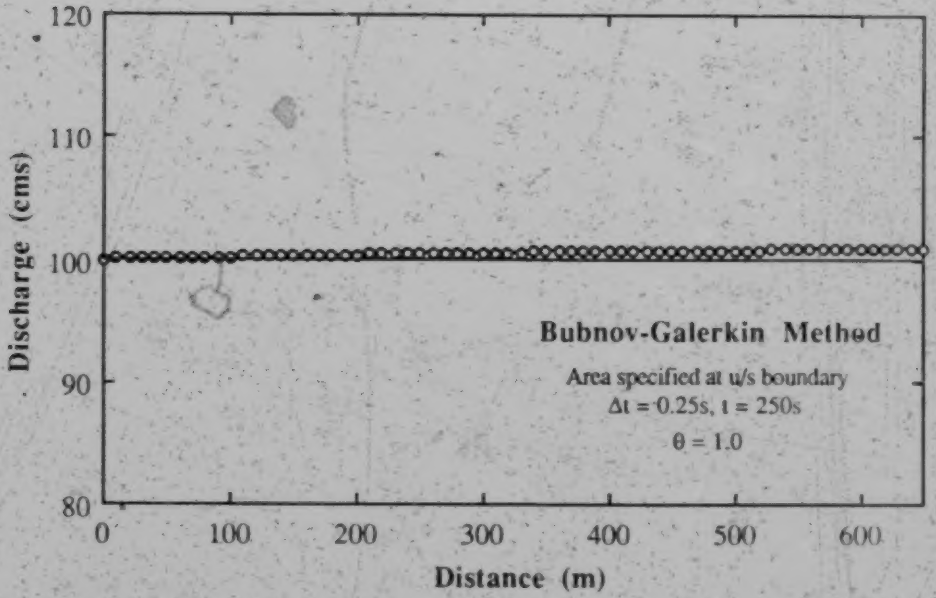
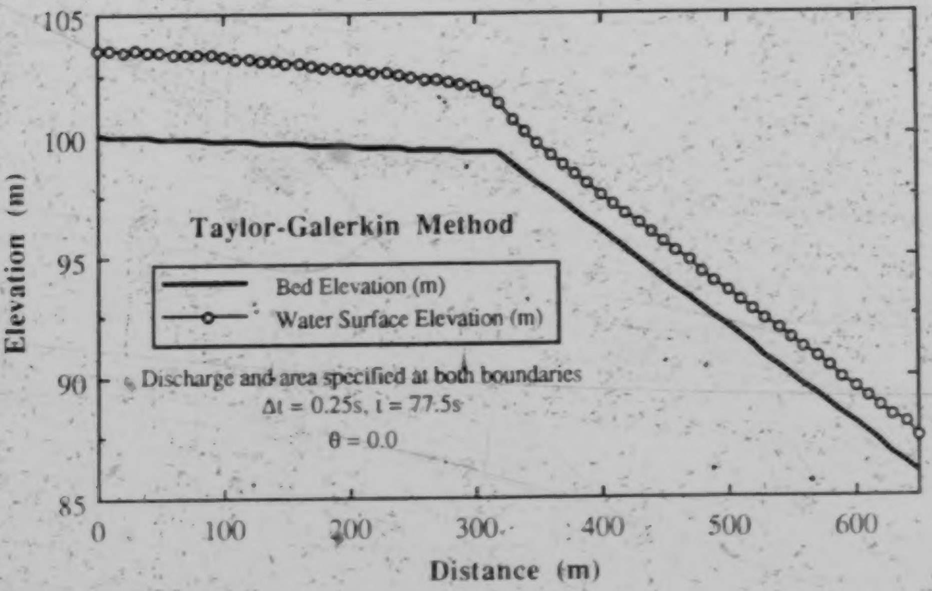


Figure 5.35 Transition from a mild to a steep channel, Bubnov-Galerkin method, area specified at upstream boundary, θ = 1.0, Δt = 0.25s 1000 time steps.



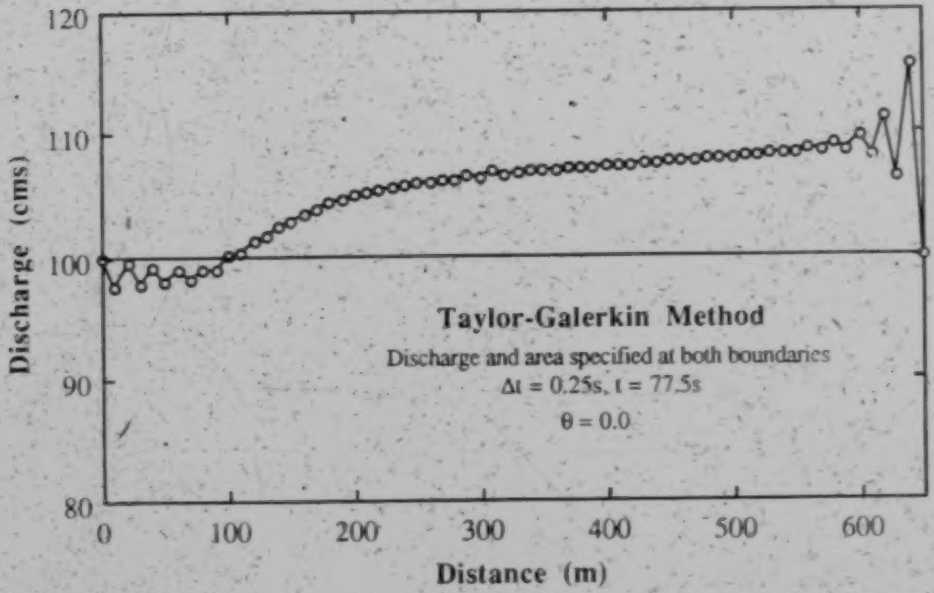
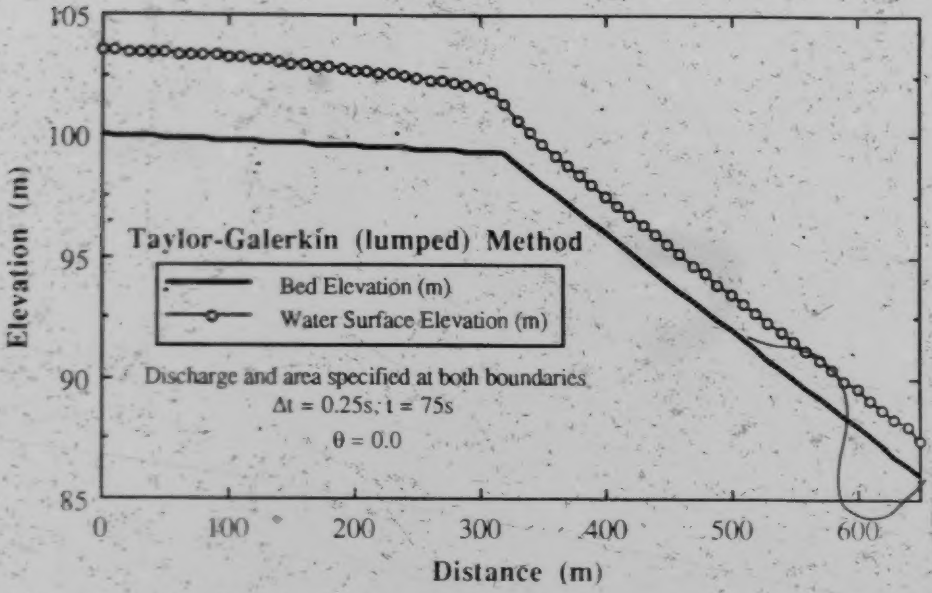


Figure 5.36 Transition from a mild to a steep channel, Taylor-Galerkin method, discharge and area specified at both boundaries, $\theta = 0.0$, $\Delta t = 0.25$s, 310 time steps.



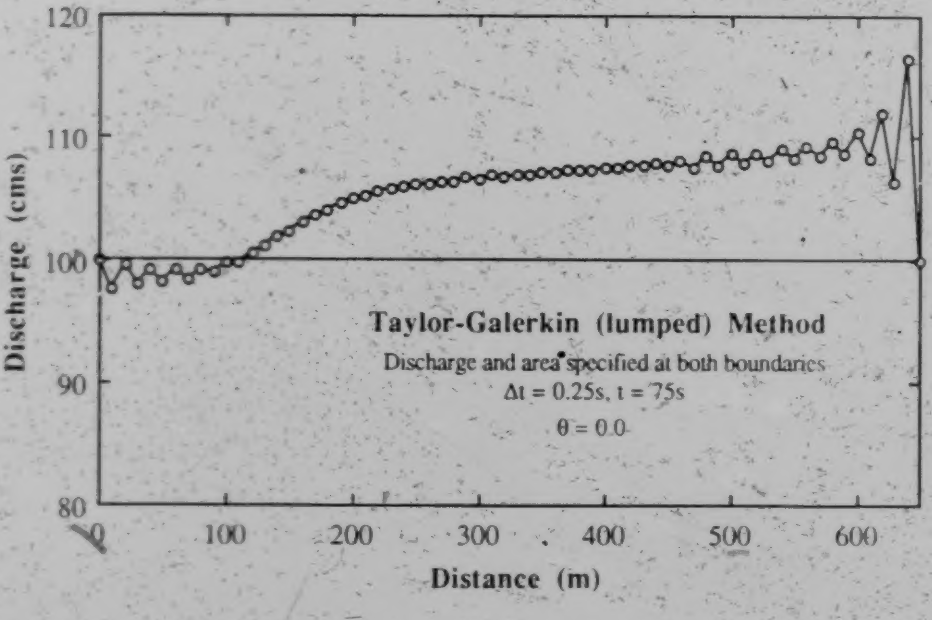
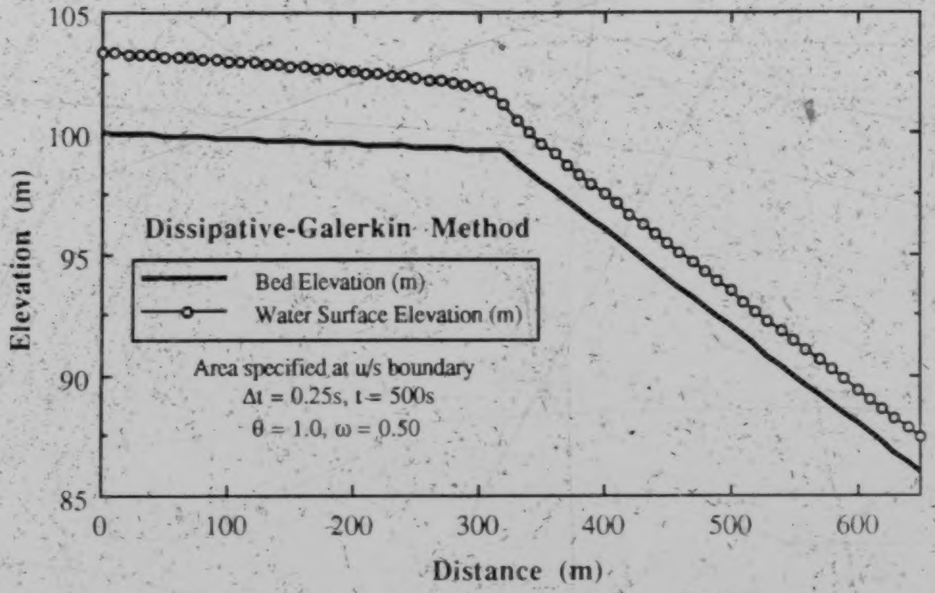


Figure 5.37 Transition from a mild to a steep channel, Taylor-Galerkin (lumped) method, discharge and area specified at both boundaries, θ = 0.0, Δt = 0.25s, 300 time steps.



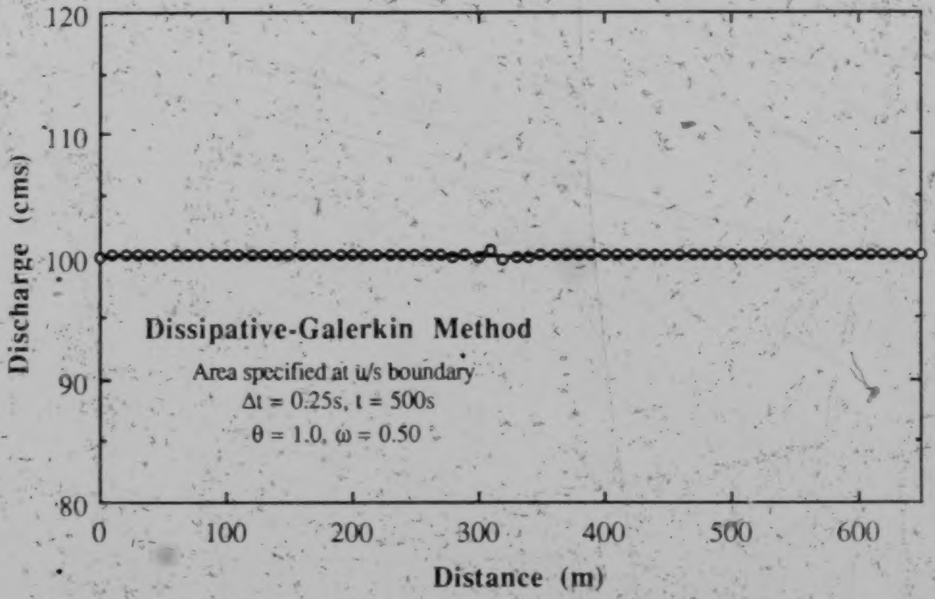


Figure 5.38 Transition from a mild to a steep channel, Dissipative-Galerkin method, area specified at upstream boundary, θ = 1.0, ω = 0.50, Δt = 0.25s, 2000 time steps.

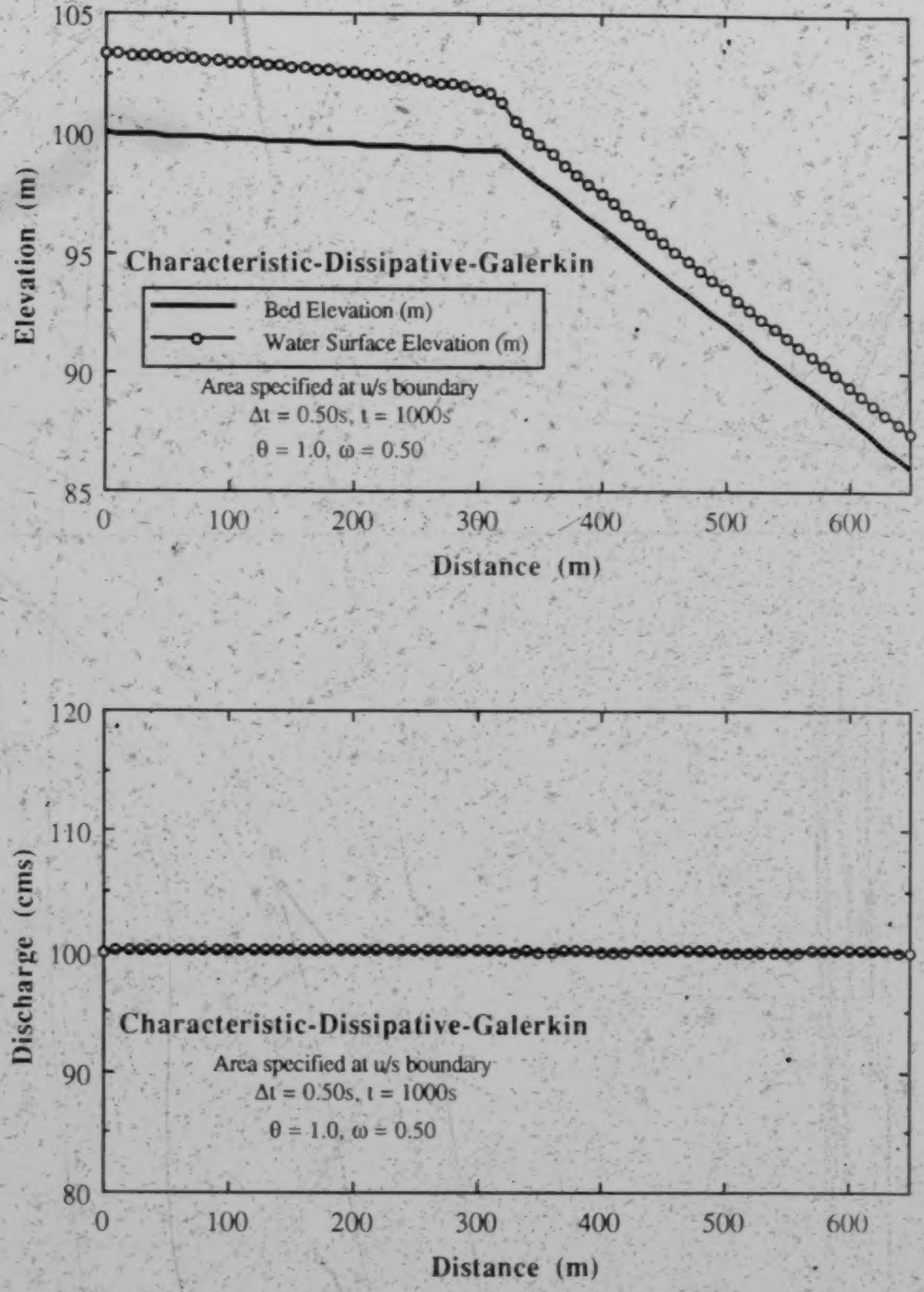
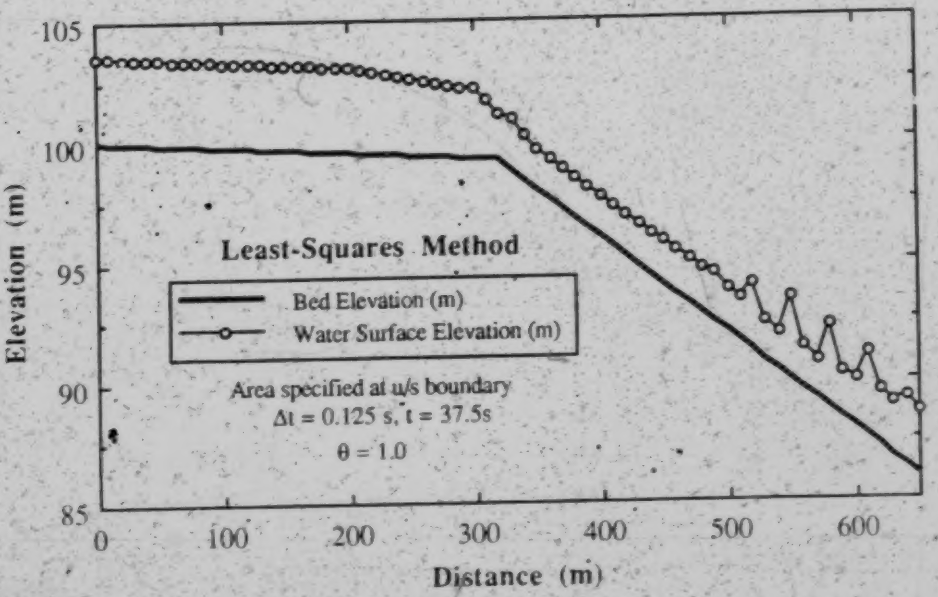


Figure 5.39 Transition from a mild to a steep channel, Characteristic-Dissipative-Galerkin method, area specified at upstream boundary, $\theta = 1.0$, $\omega = 0.50$, $\Delta t = 0.5$s, 2000 time steps.



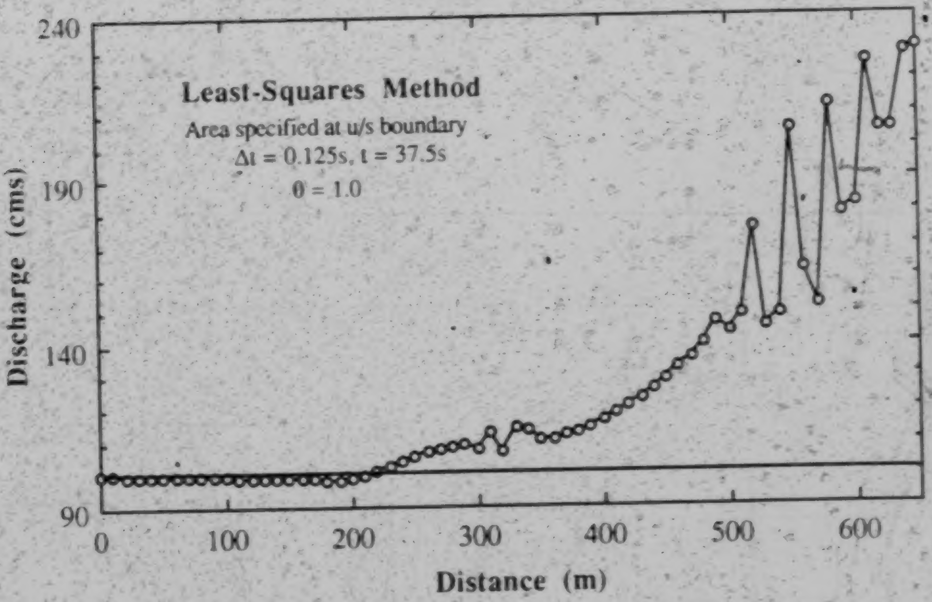


Figure 5.40 Transition from a mild to a steep channel, Least-Squares method, area specified at upstream boundary, $\theta = 1.0$, $\Delta t = 0.125$s, 300 time steps.

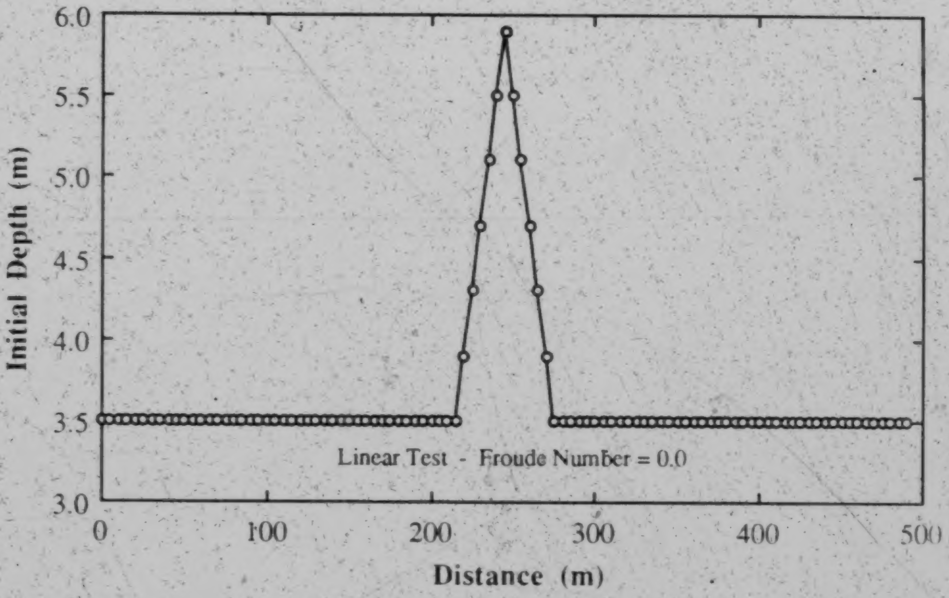
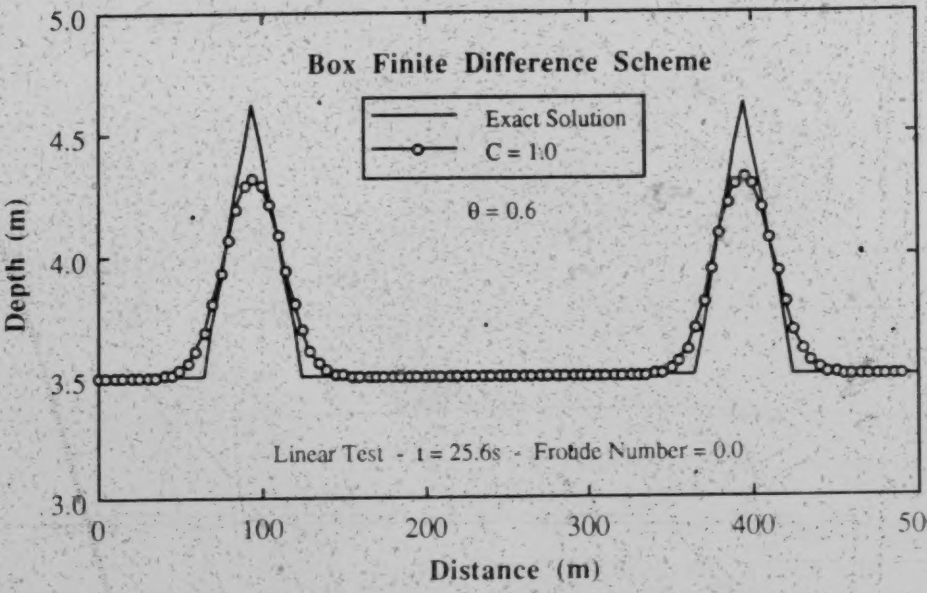
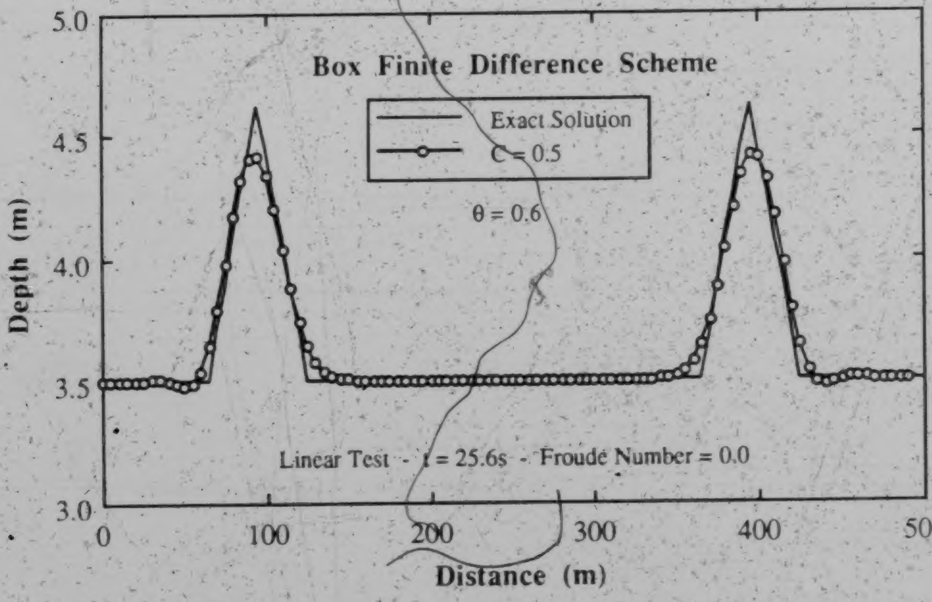


Figure 5.41 Initial conditions for linear test, **F** = 0.0.



(a)



(b)

Figure 5.42 Linear perturbation (t = 25.6s), Box Finite Difference method, F = 0.0, $\theta = 0.5$, (a) C = 1.0, $\Delta t = 0.8533s$; (b) C = 0.5, $\Delta t = 0.42665s$

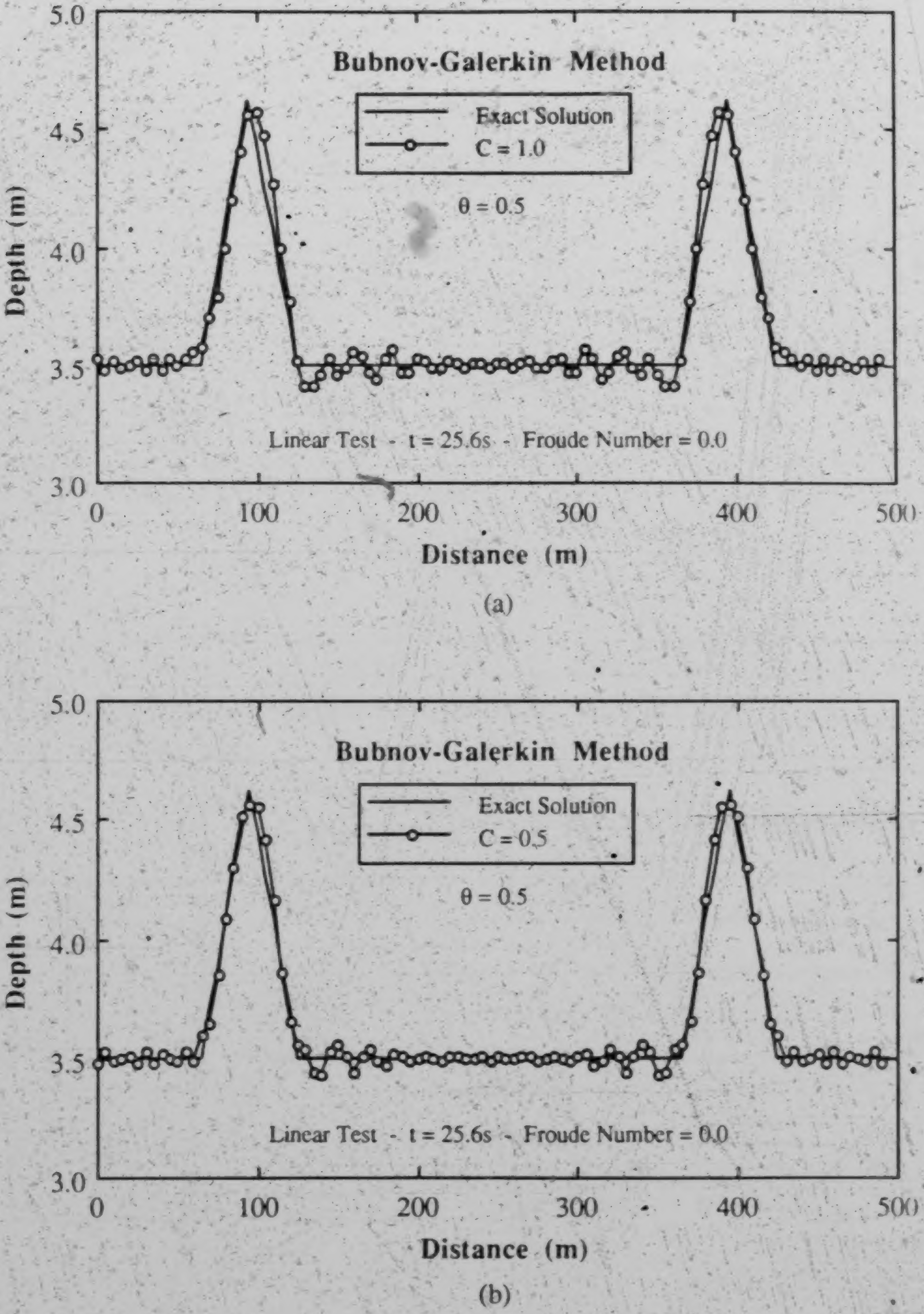
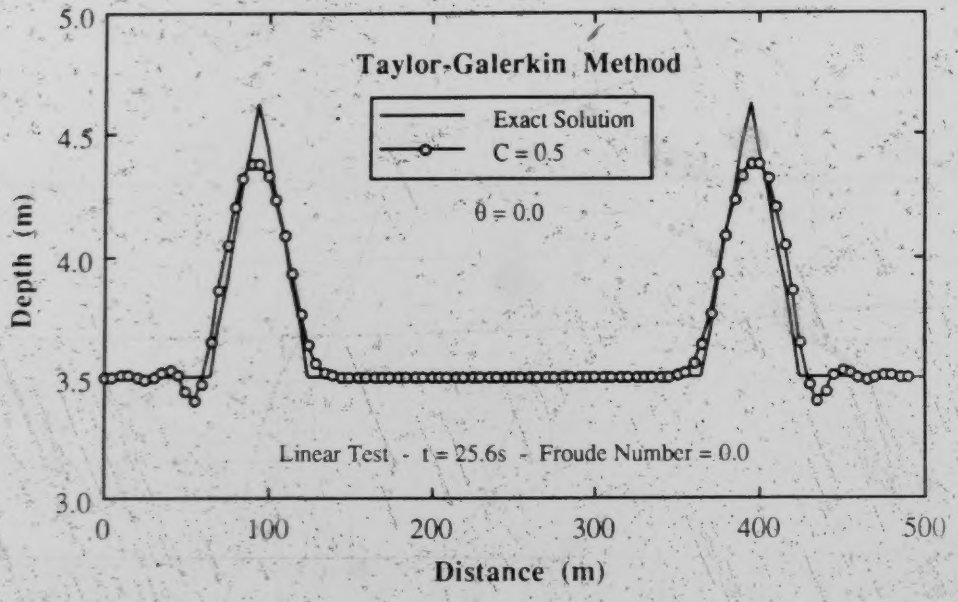
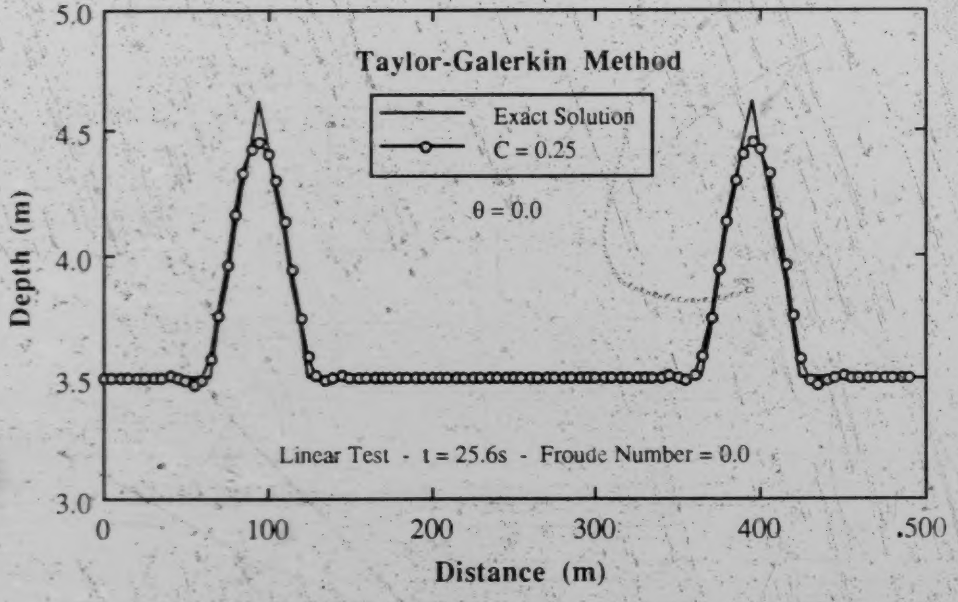


Figure 5.43 Linear perturbation (t = 25.6s), Bubnov-Galerkin method, **F** = 0.0, $\theta = 0.5$ (a) C = 1.0, $\Delta t = 0.8533$s; (b) C = 0.5, $\Delta t = 0.42665$s.

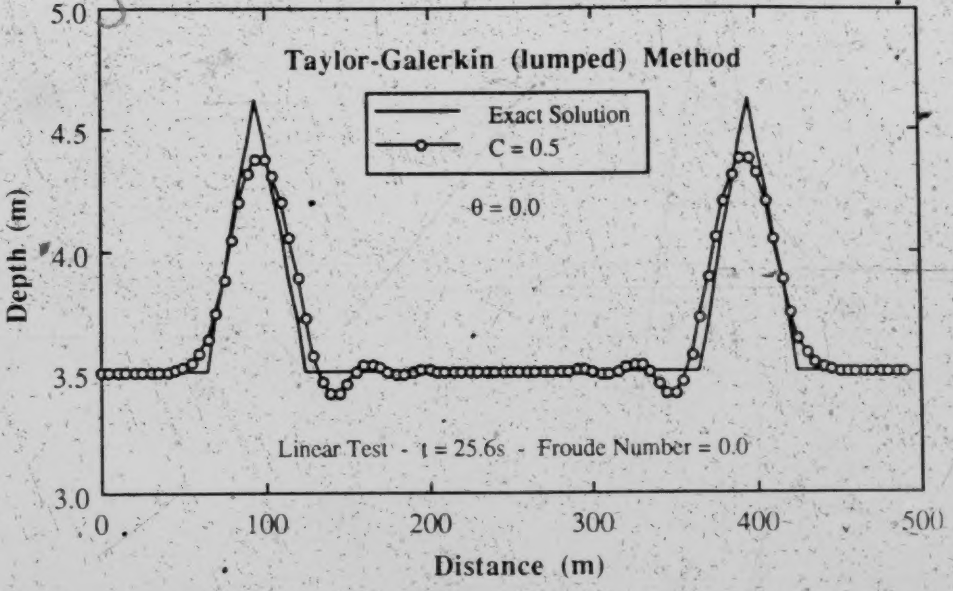


(a)

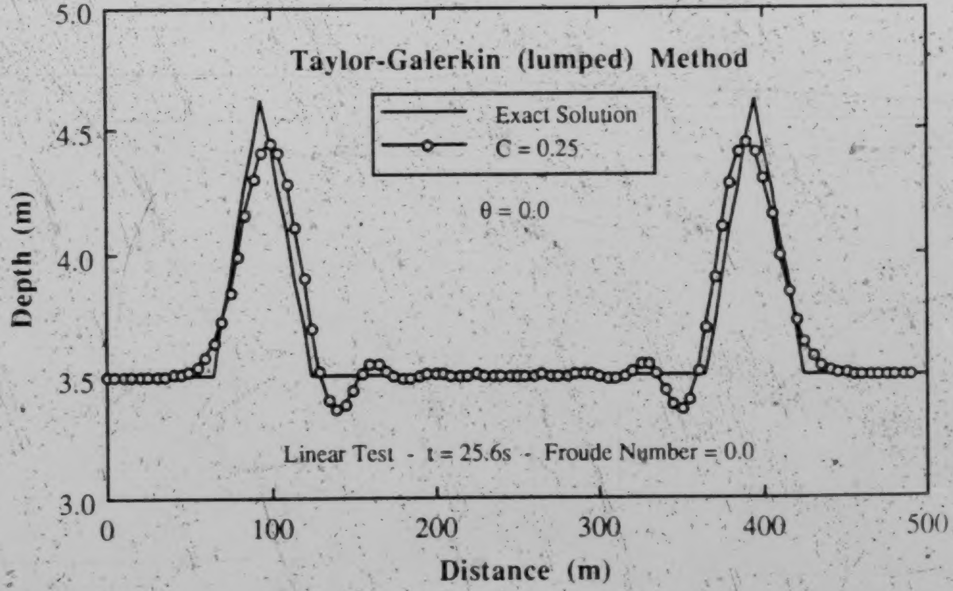


(b)

Figure 5.44 Linear perturbation (t = 25.6s), Taylor-Galerkin method, **F** = 0.0, θ = 0.0, (a) C = 0.5, Δt = 0.42665s; (b) C = 0.25, Δt = 0.213325s.

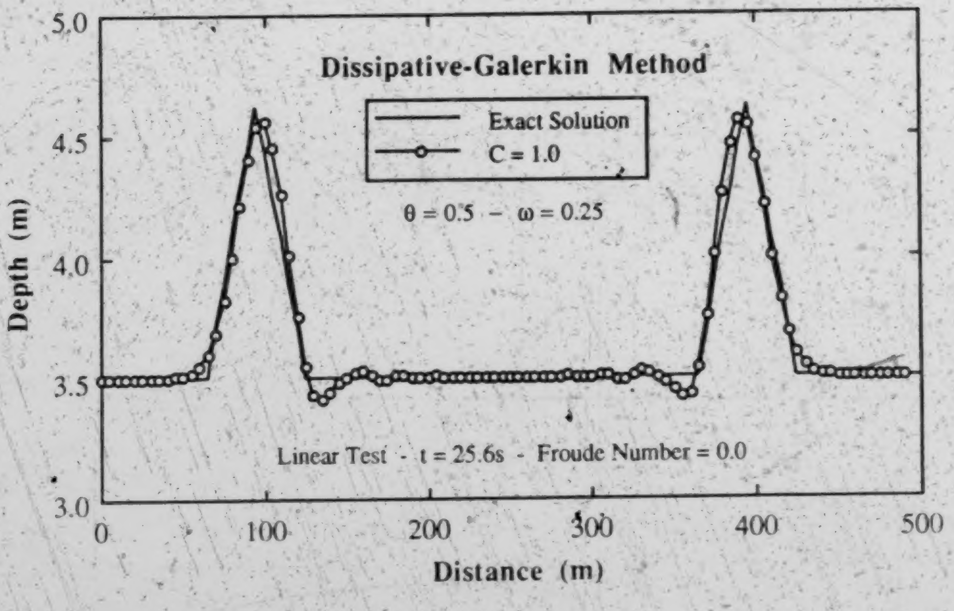


(a)

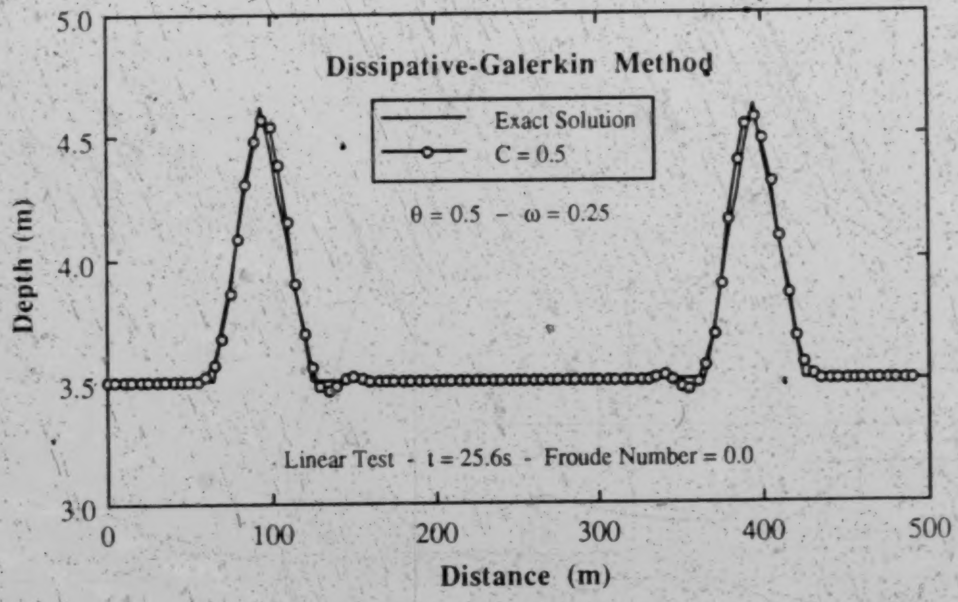


(b)

Figure 5.451 Linear perturbation ($t = 25.6s$), Taylor-Galerkin (lumped) method; $\mathbf{F} = 0.0$, $\theta = 0.0$ (a) $C = 0.5$, $\Delta t = 0.42665s$; (b) $C = 0.25$, $\Delta t = 0.213325s$.

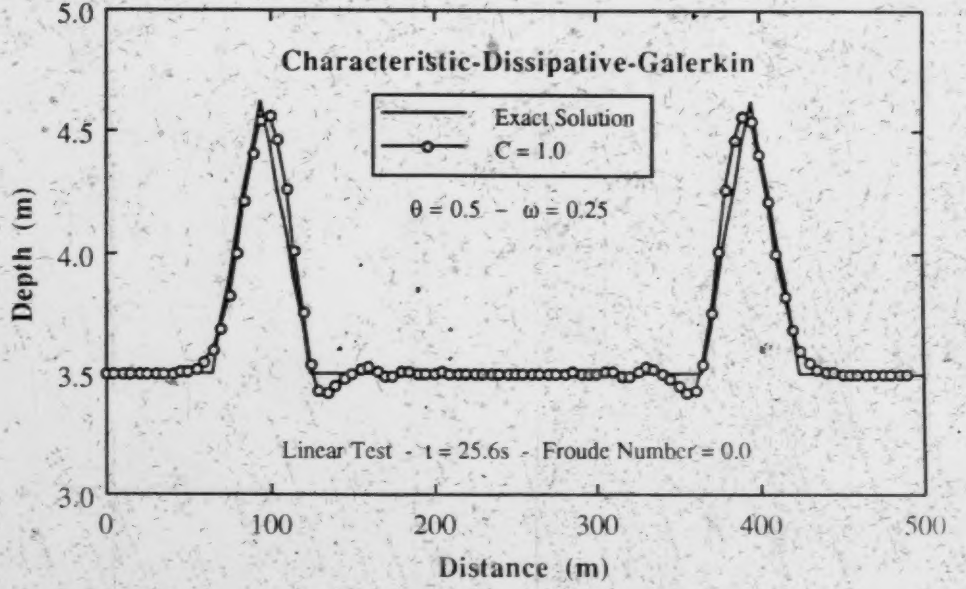


(a)

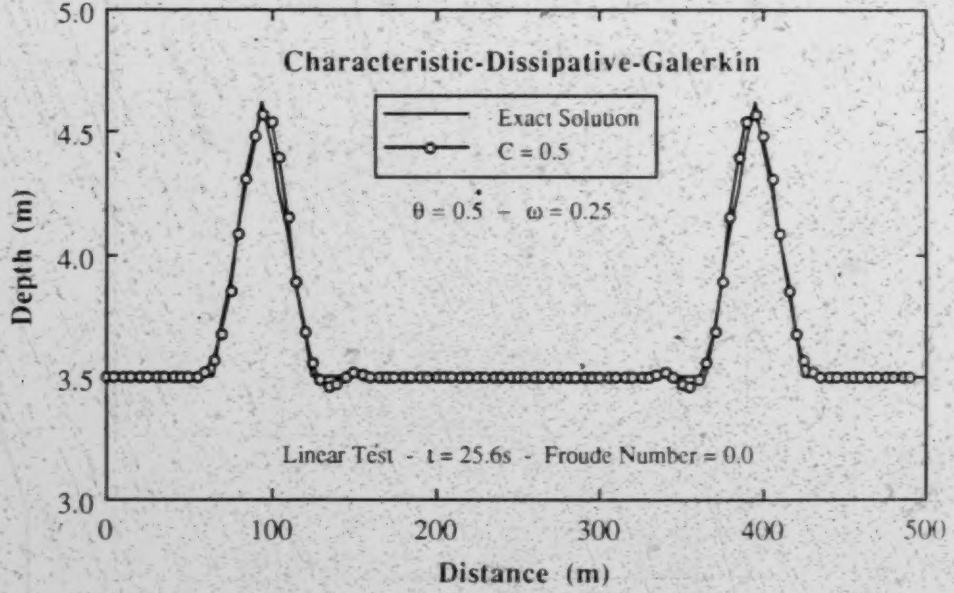


(b)

Figure 5.46 Linear perturbation (t = 25.6s), Dissipative-Galerkin method, F = 0.0, θ = 0.5, ω = 0.25, (a) C = 1.0, Δt = 0.8533s; (b) C = 0.5, Δt = 0.42665s.

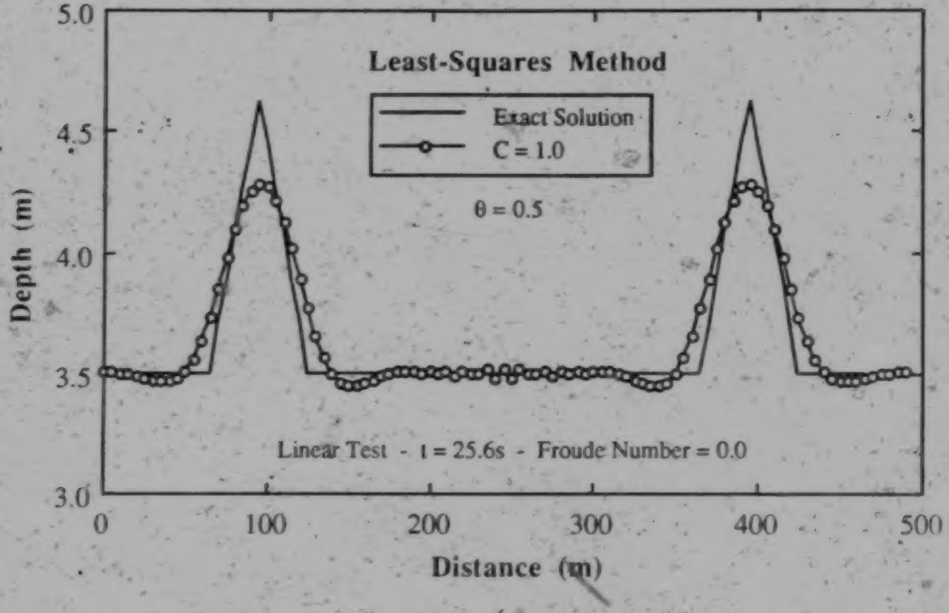


(a)

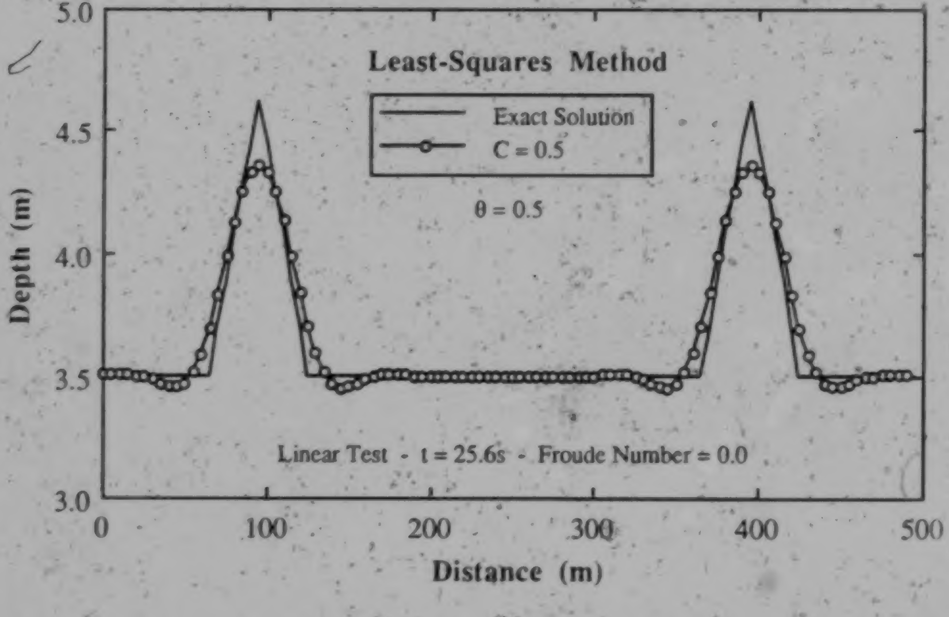


(b)

Figure 5.47 Linear perturbation (t = 25.6s), Characteristic-Dissipative-Galerkin method, $\mathbf{F} = 0.0$, $\theta = 0.5$, $\omega = 0.25$, (a) $C = 1.0$, $\Delta t = 0.8533s$; (b) $C = 0.5$, $\Delta t = 0.42665s$.

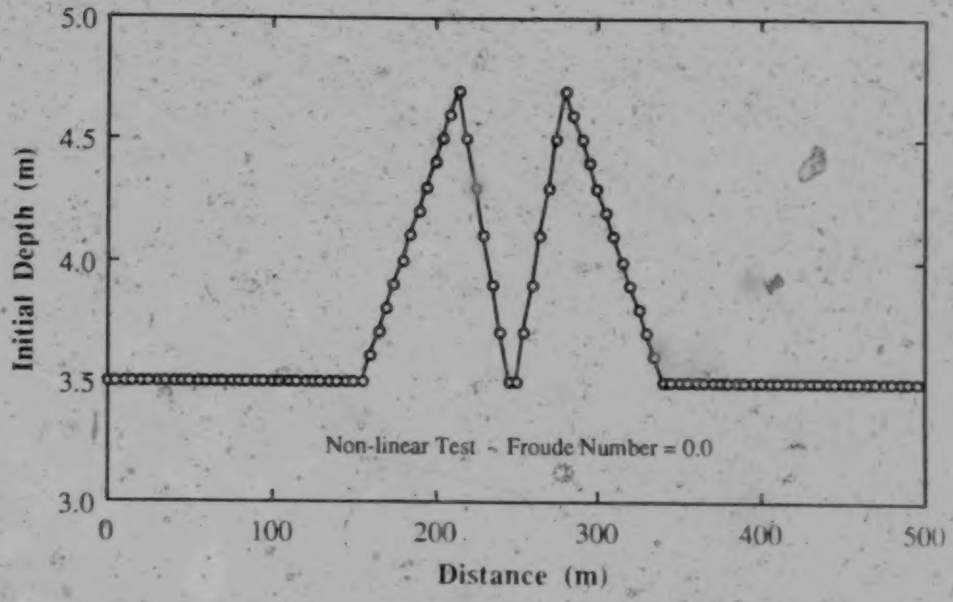


(a)



(b)

Figure 5.48 Linear perturbation (t = 25.6s), Least-Squares method, F = 0.0, θ = 0.5, (a) C = 1.0, Δt = 0.8533s; (b) C = 0.5, Δt = 0.42665s.



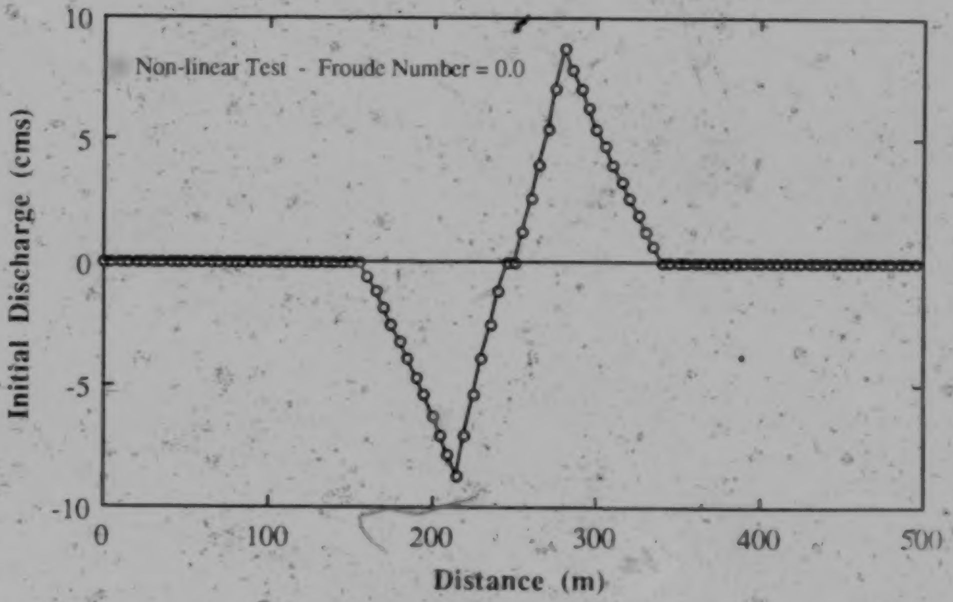
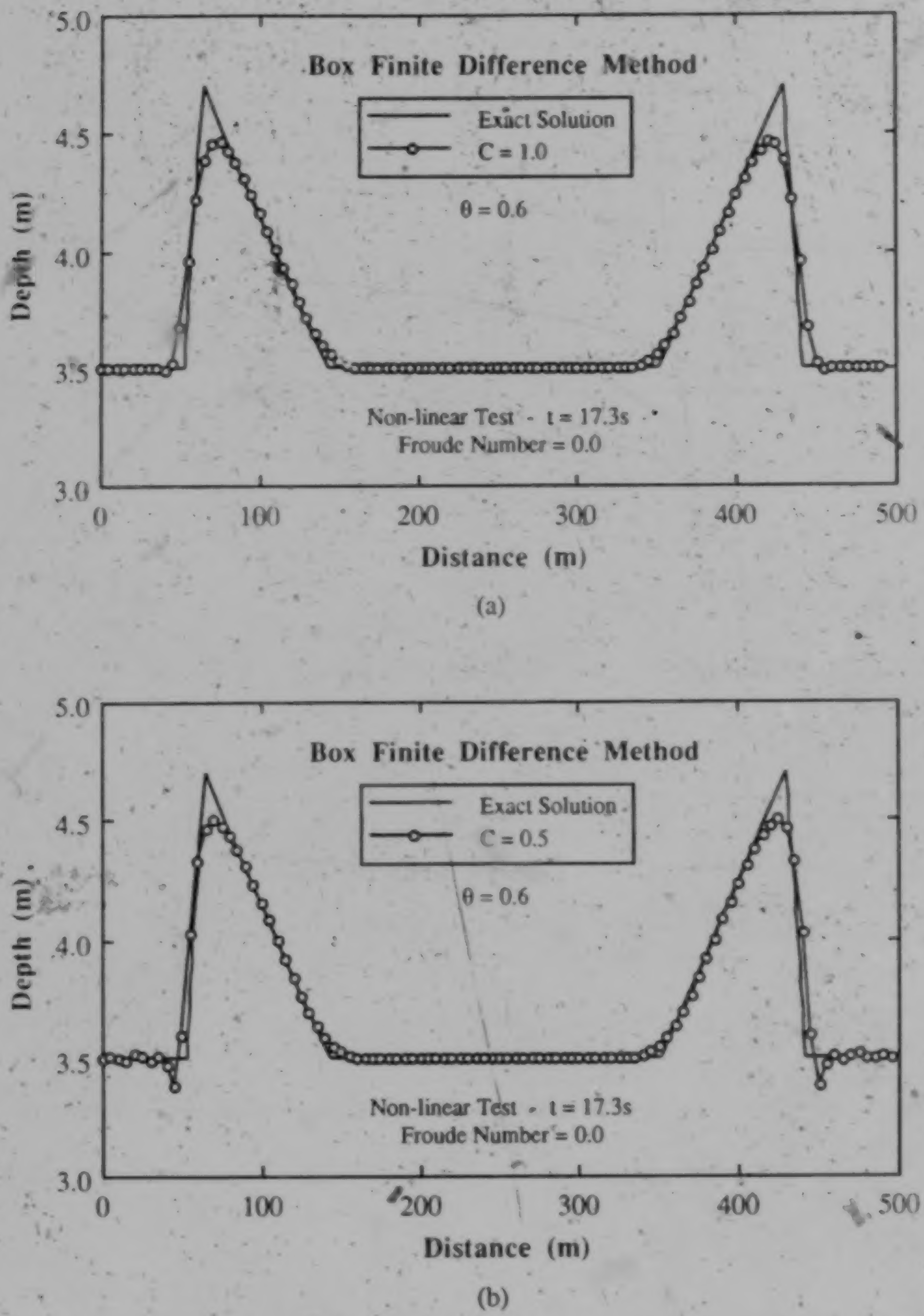


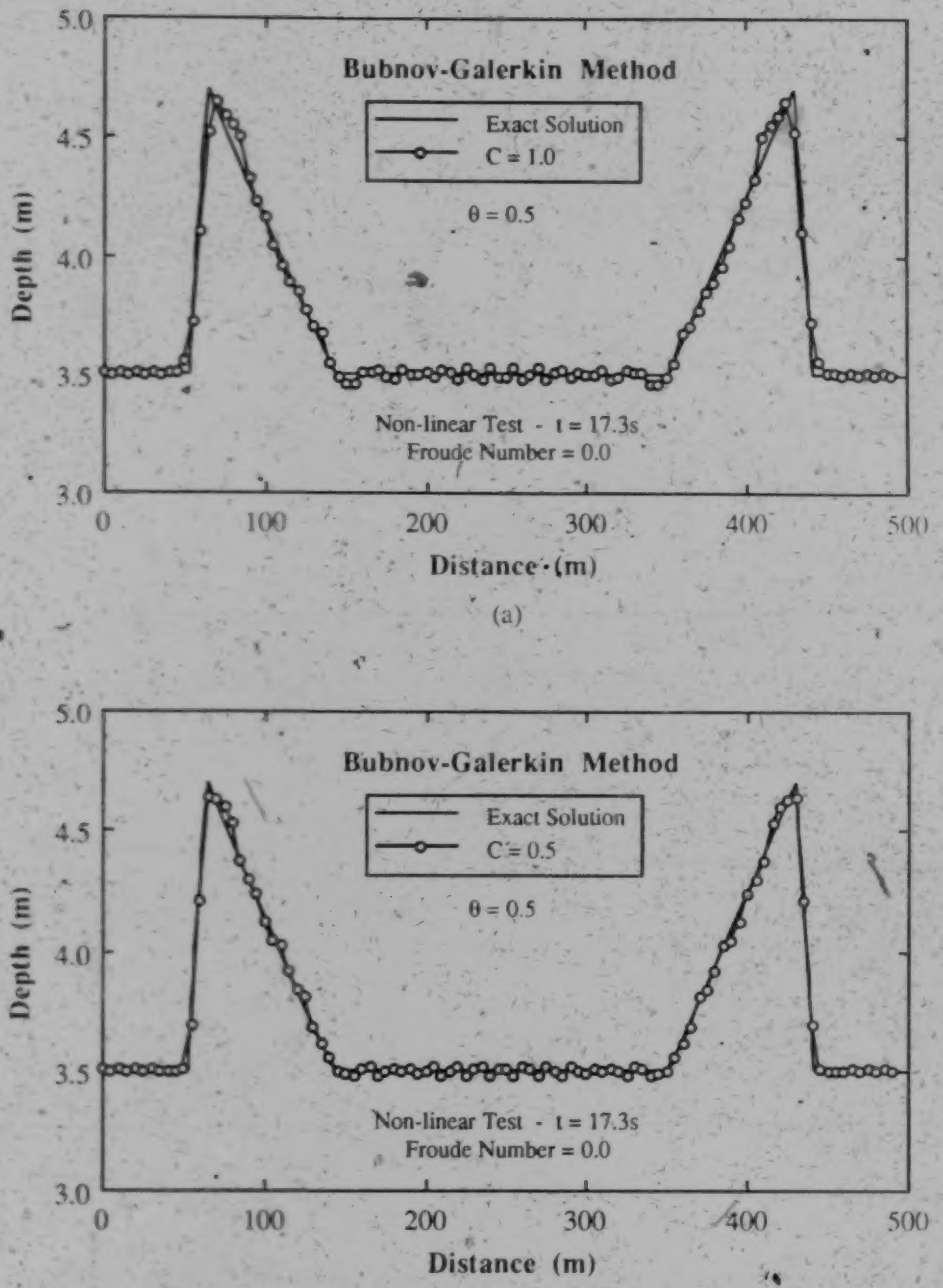
Figure 5.49 Initial conditions for non-linear test, **F** = 0.0.



(a)

(b)

Figure 5.50 Non-linear perturbation (t = 17.3s), Box Finite Difference method, F = 0.0, θ = 0.6, (a) C = 1.0, Δt = 0.5780s; (b) C = 0.5, Δt = 0.2890s.



(a)

(b)

Figure 5.51 Non-linear perturbation (t = 17.3s), Bubnov-Galerkin method, F = 0.0, θ = 0.5 (a) C = 1.0, Δt = 0.5780s; (b) C = 0.5, Δt = 0.2890s.

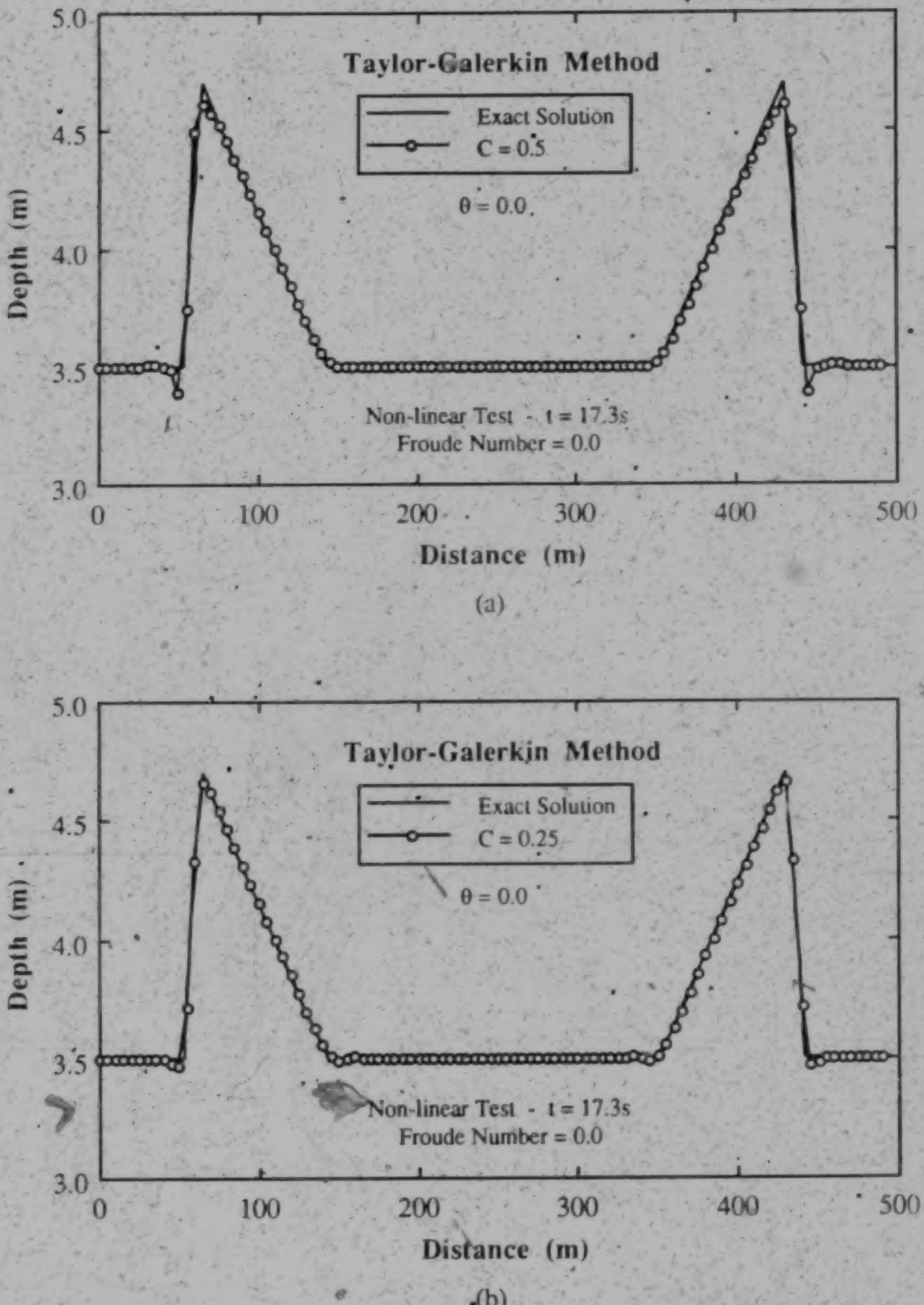
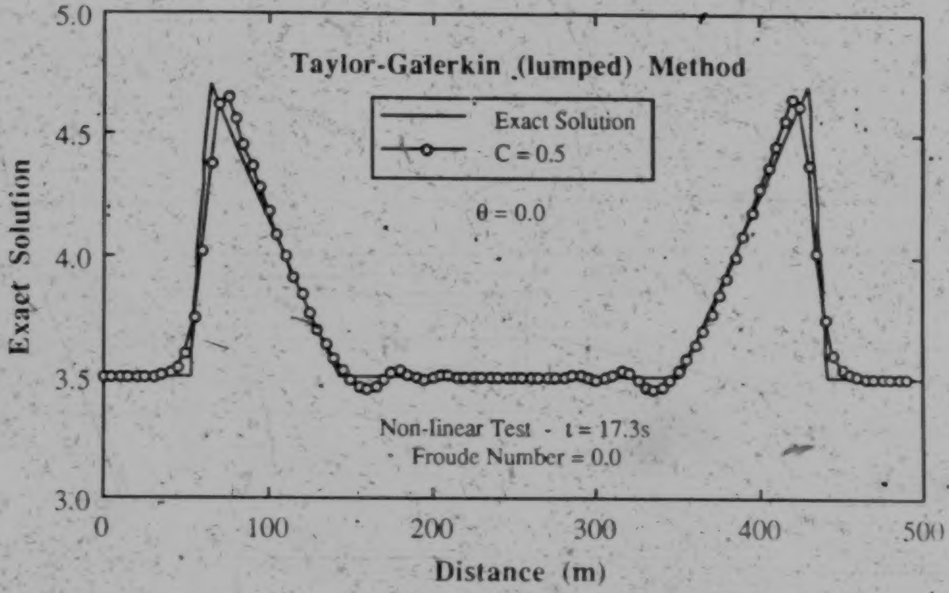
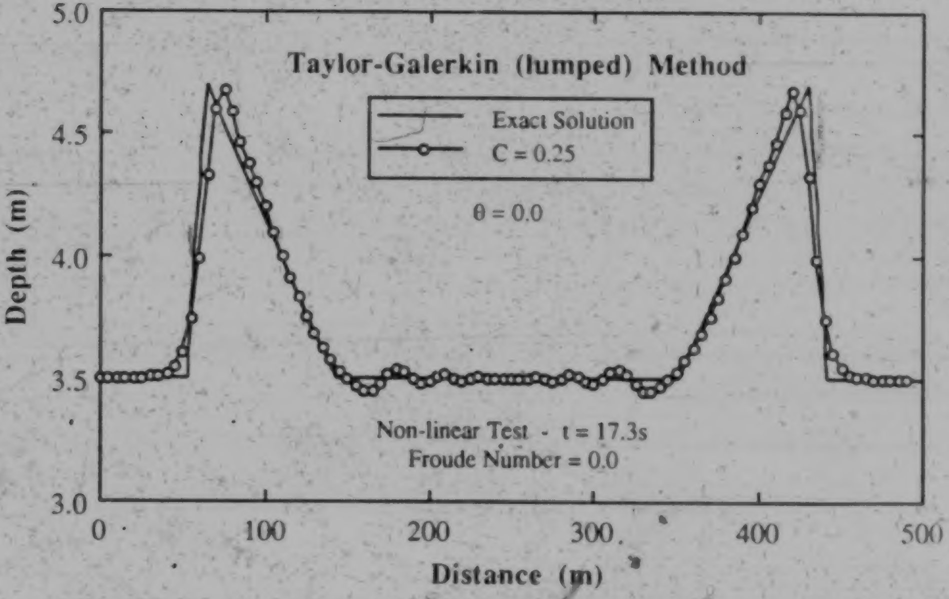


Figure 5.52 Non-linear perturbation (t = 17.3s), Taylor-Galerkin method, **F** = 0.0, θ = 0.0, (a) C = 0.5, Δt = 0.2890s; (b) C = 0.25, Δt = 0.1445s.



(a)



(b)

Figure 5.53 Non-linear perturbation (t = 17.3s), Taylor-Galerkin (lumped) method, F = 0.0, θ = 0.0 (a) C = 0.5,. Δt = 0.2890s; (b) C = 0.25, Δt = 0.1445s.

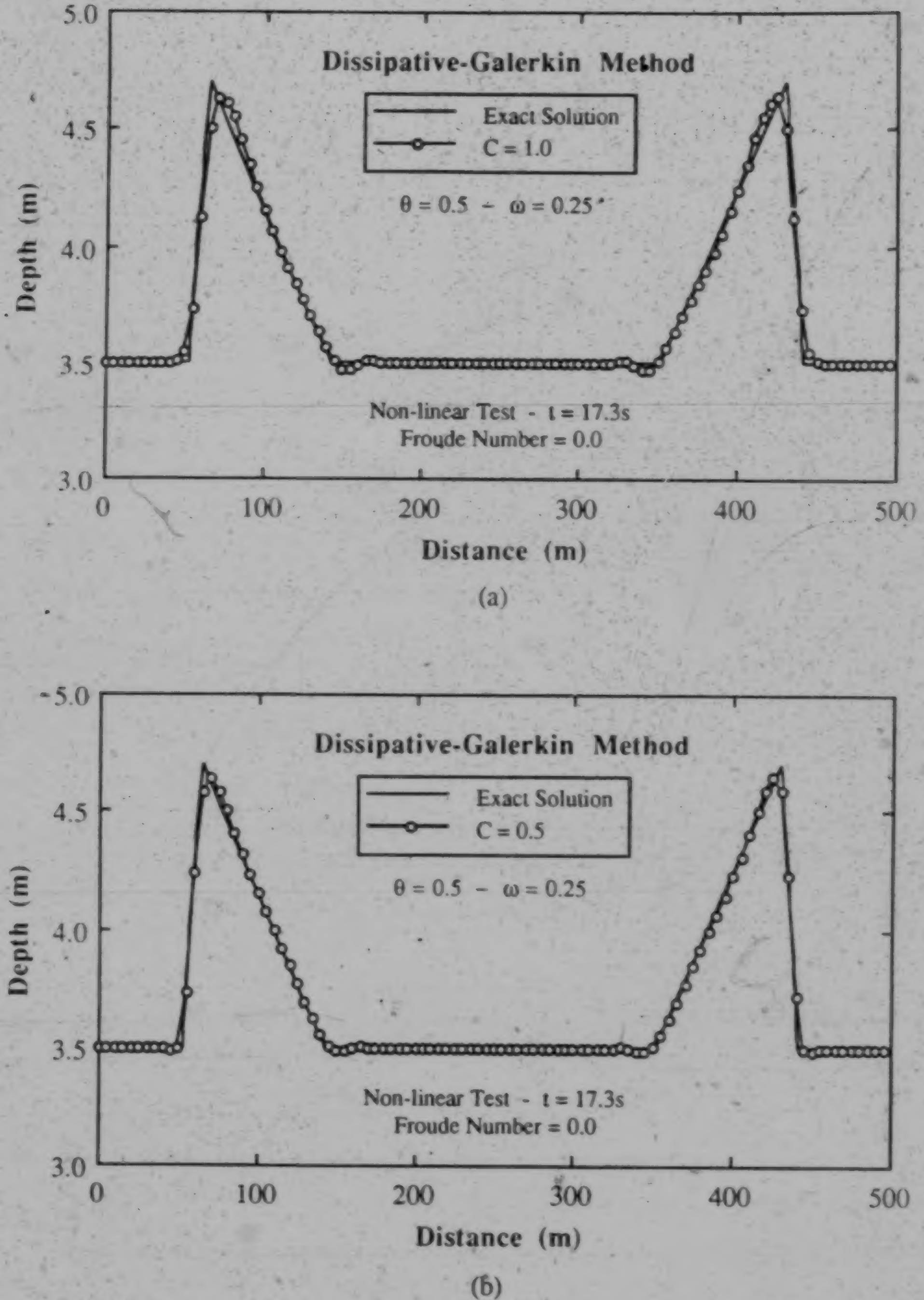
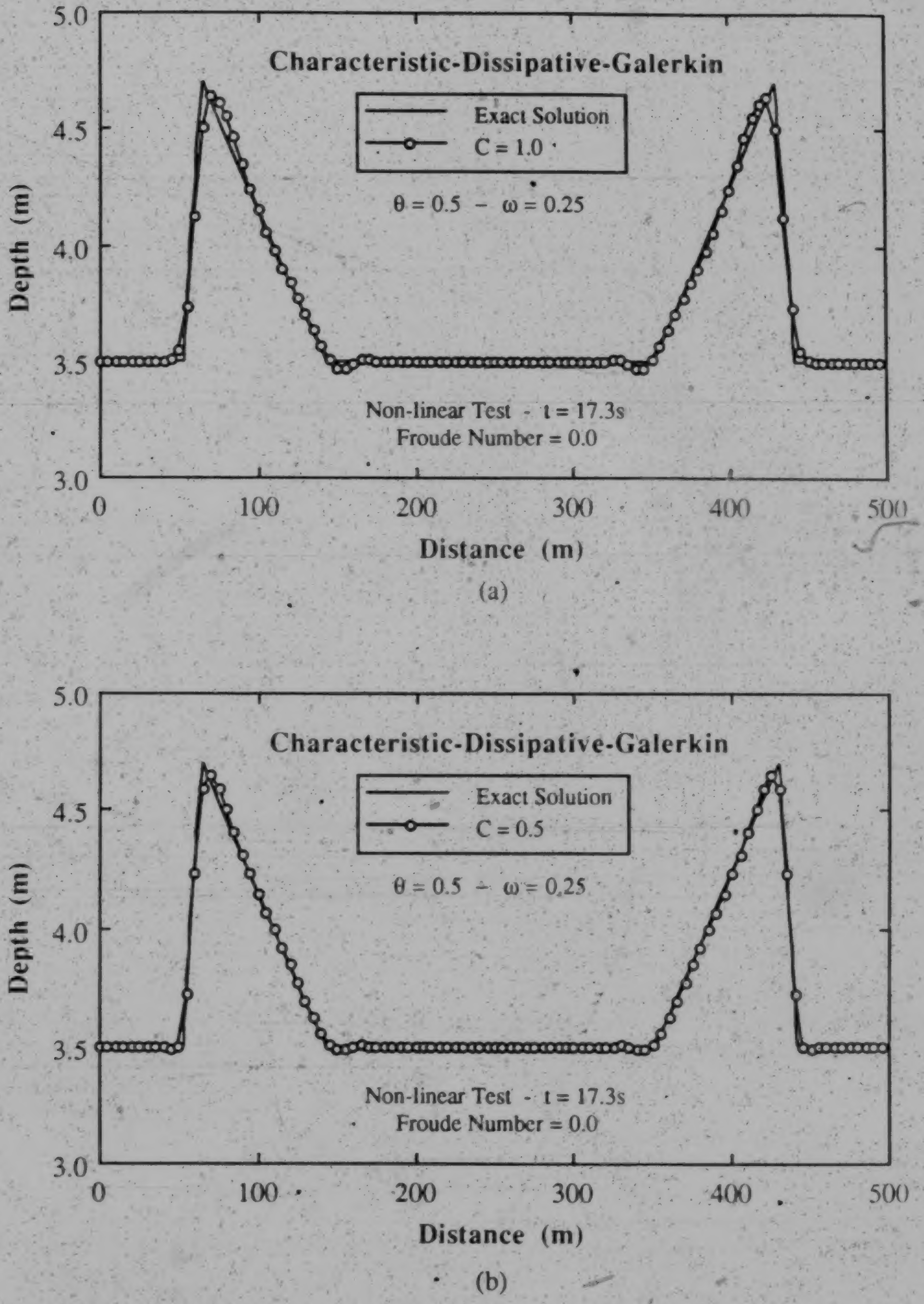


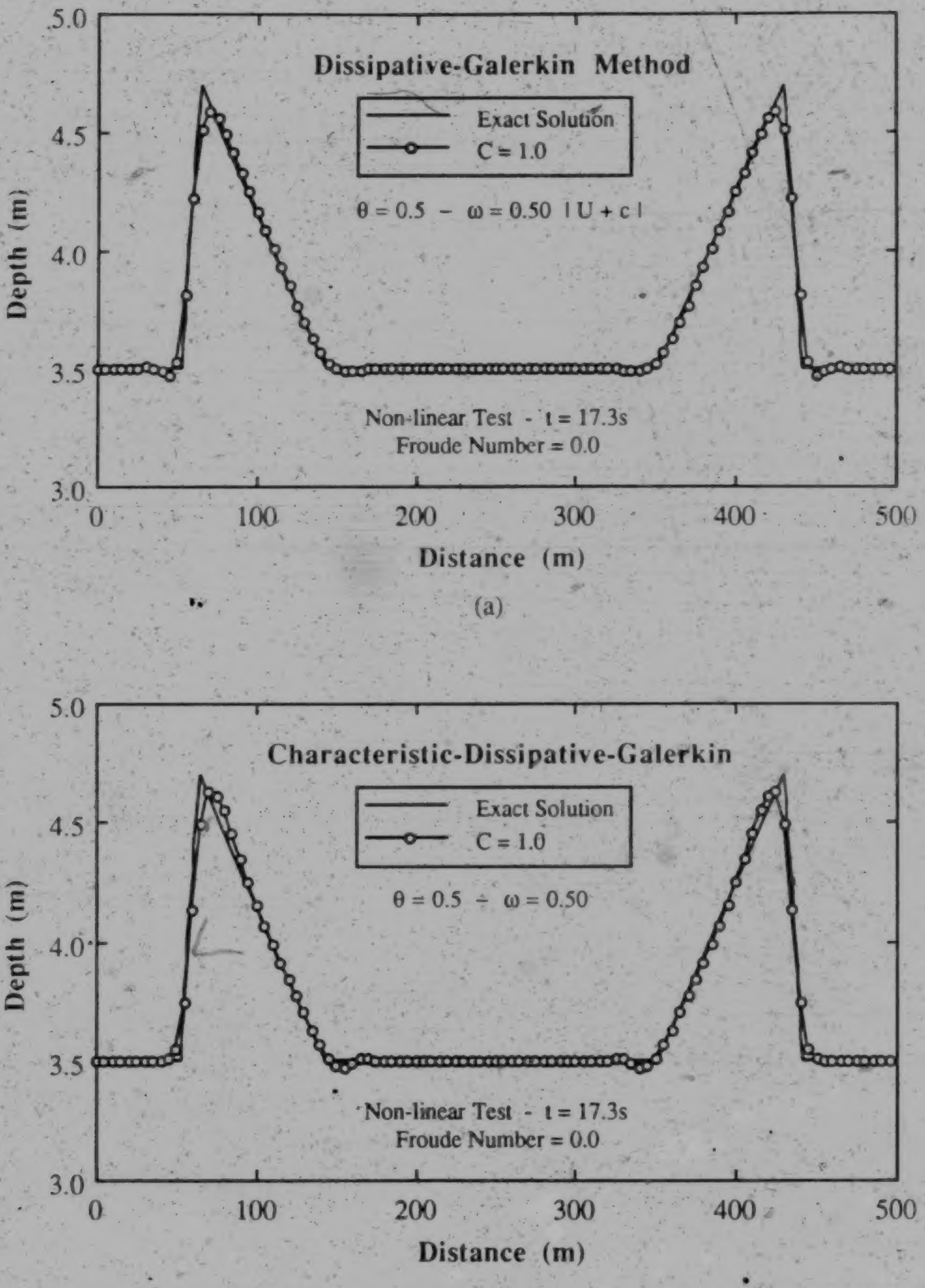
Figure 5.54 Non-linear perturbation ($t = 17.3s$), Dissipative-Galerkin method, $F = 0.0$, $\theta = 0.5$, $\omega = 0.25$, (a) $C = 1.0$, $\Delta t = 0.5780s$; (b) $C = 0.5$, $\Delta t = 0.2890s$.



(a)

(b)

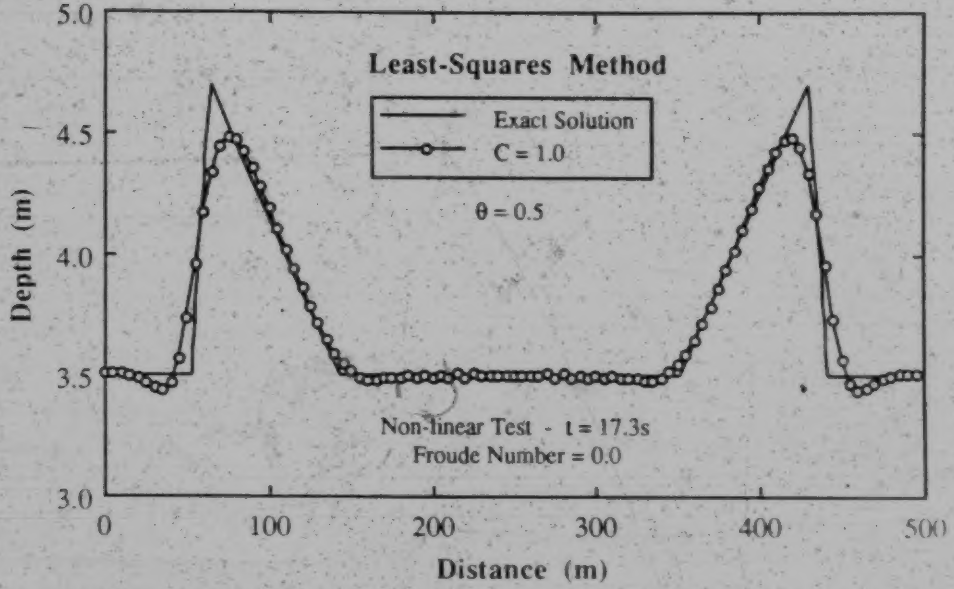
Figure 5.55 Non-linear perturbation (t = 17.3s), Characteristic-Dissipative-Galerkin method, **F** = 0.0, θ = 0.5, ω = 0.25, (a) C = 1.0, Δt = 0.5780s; (b) C = 0.5, Δt = 0.2[illegible]s.



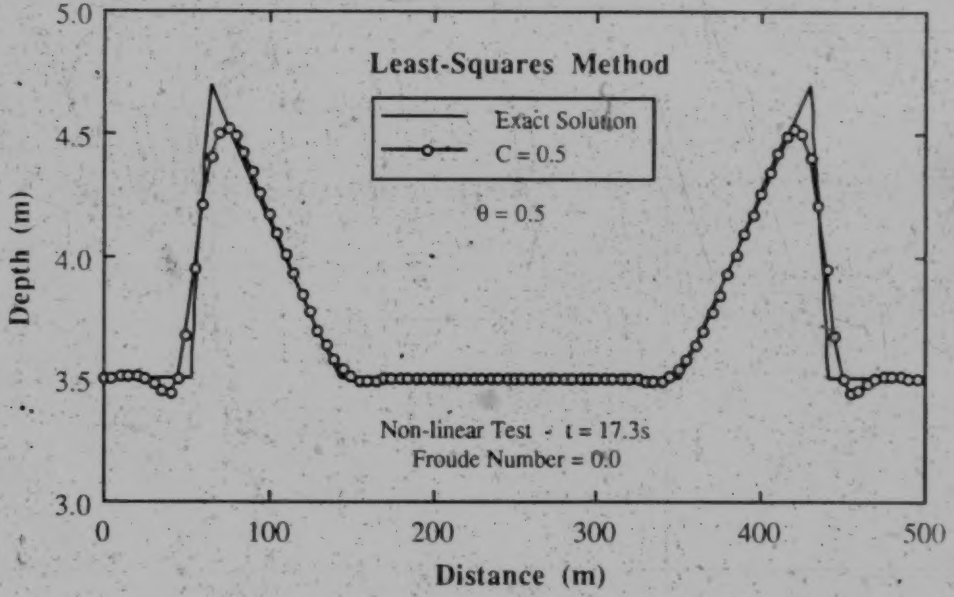
(a)

(b)

Figure 5.56 Non-linear perturbation (t = 17.3s), **F** = 0.0, θ = 0.5, C = 1.0, Δt = 0.5780s, (a) Dissipative-Galerkin method, ω = 0.50 | U + c |; (b) Characteristic-Dissipative-Galerkin method, ω = 0.50.

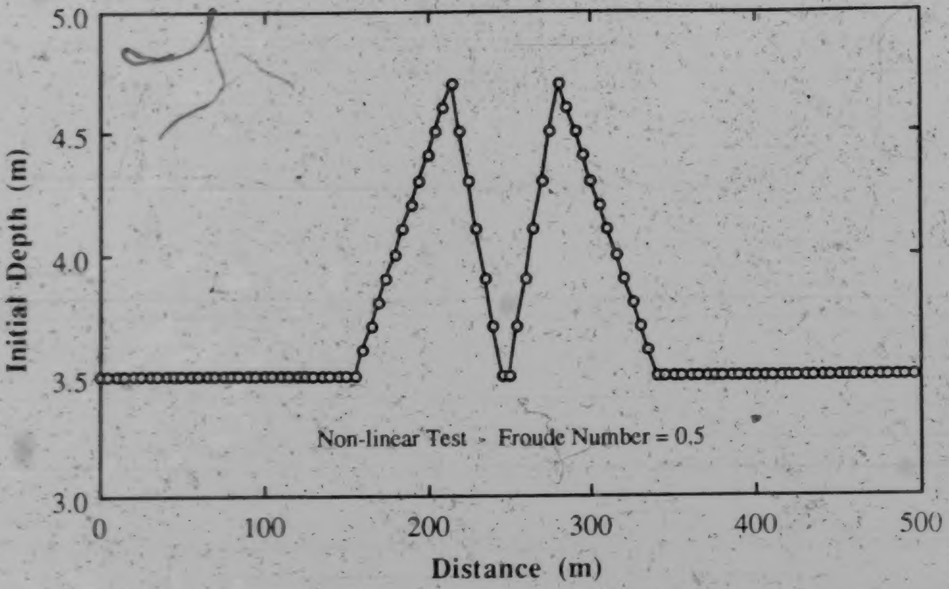


(a)



(b)

Figure 5.57 Non-linear perturbation (t = 17.3s), Least-Squares method, F = 0.0, $\theta = 0.5$, (a) C = 1.0, $\Delta t = 0.5780$s; (b) C = 0.5, $\Delta t = 0.2890$s.



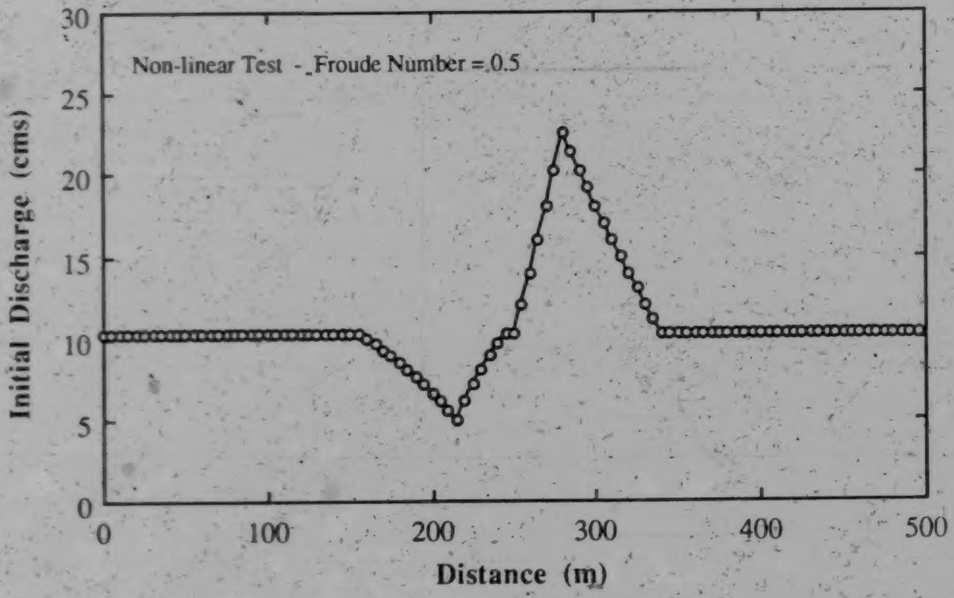
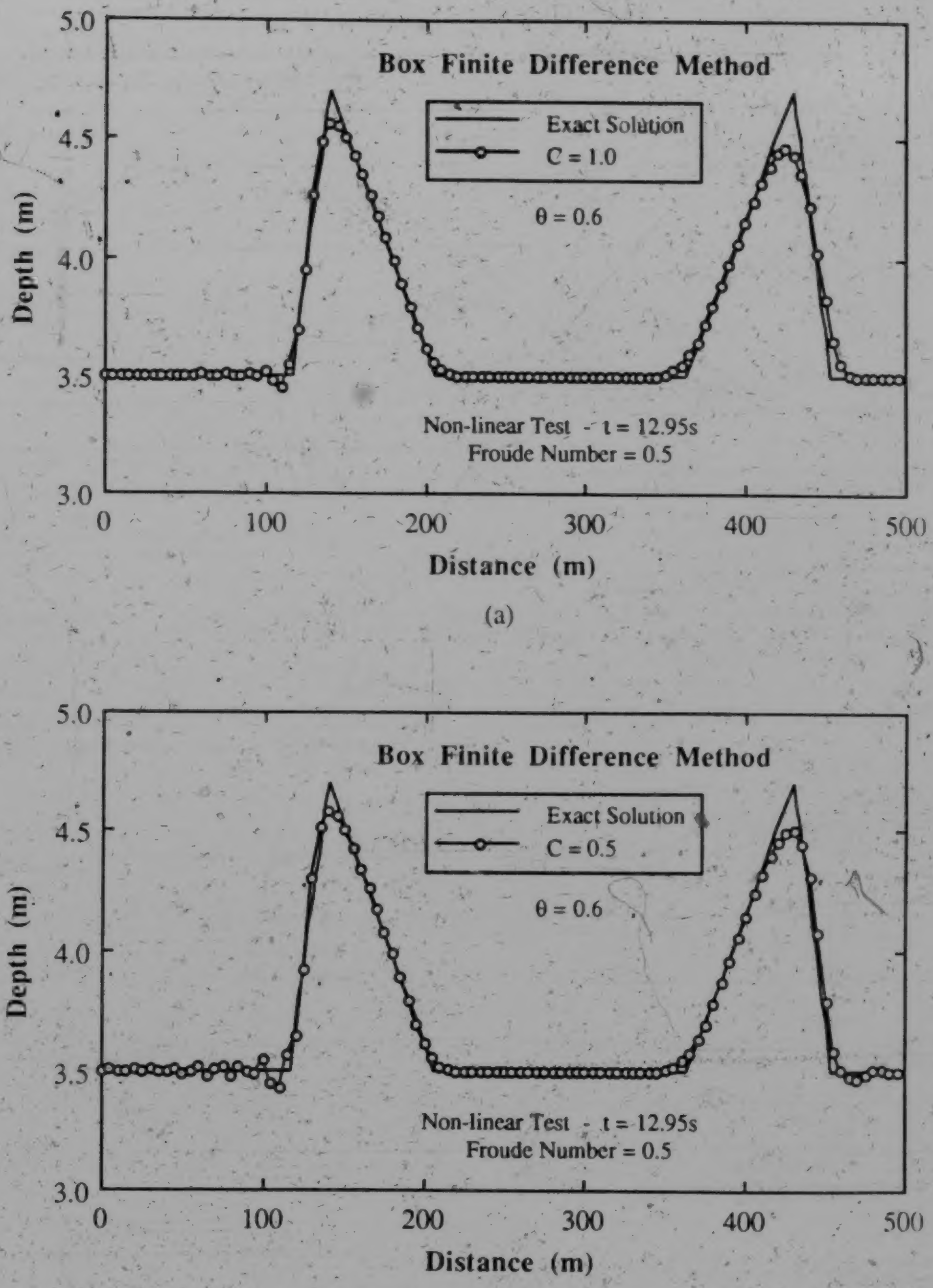


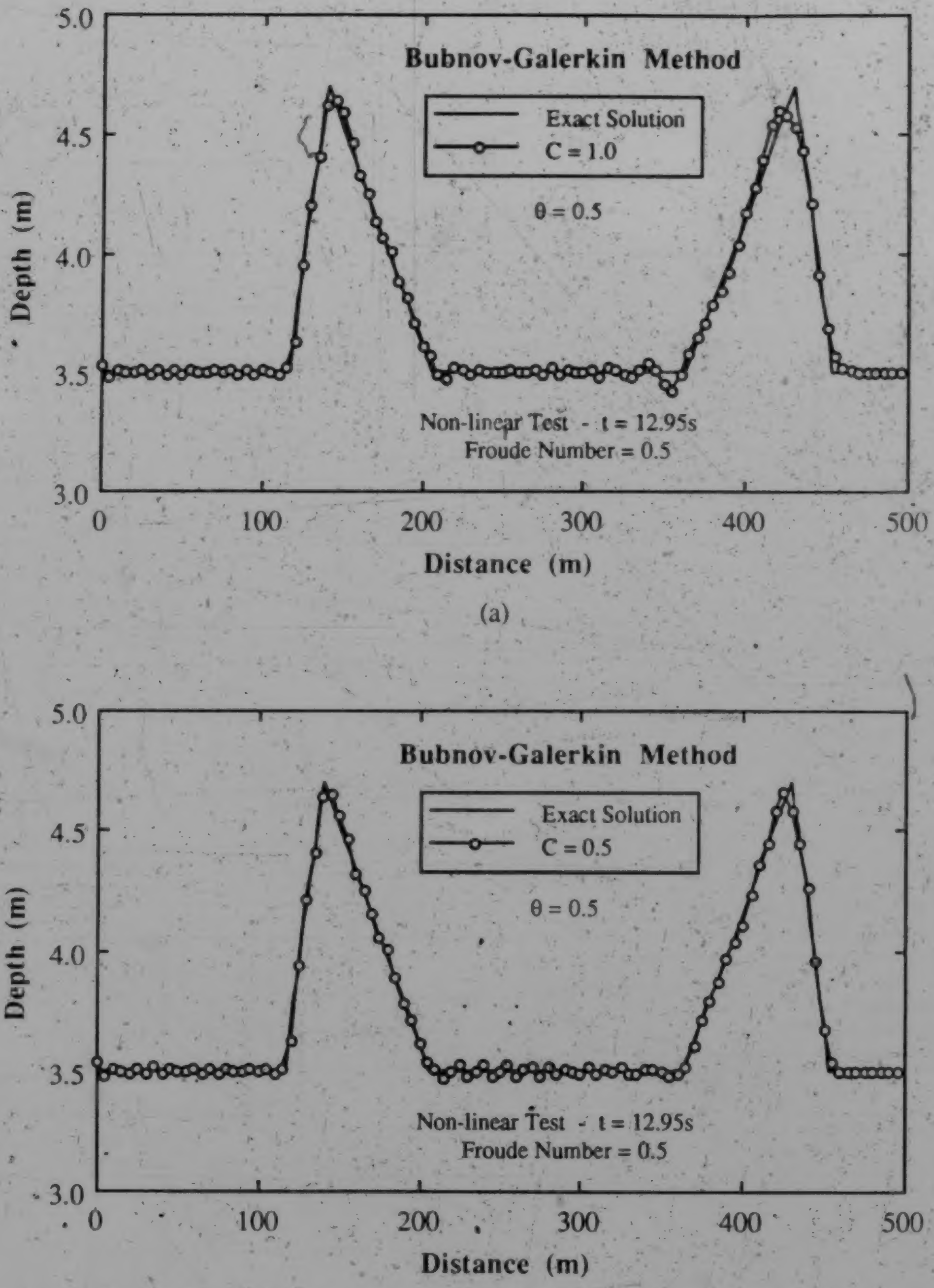
Figure 5.58 Initial conditions for non-linear test, **F** = 0.5.



(a)

(b)

Figure 5.59 Non-linear perturbation (t = 12.95s), Box Finite Difference Method, F = 0.5, θ = 0.6, (a) C = 1.0, Δt = 0.4317s; (b) C = 0.5, Δt = 0.2159s.



(a)

(b)

Figure 5.60 Non-linear perturbation (t = 12.95s), Bubnov-Galerkin method, **F** = 0.5, θ = 0.5 (a) C = 1.0, Δt = 0.4317s; (b) C = 0.5, Δt = 0.2159s.

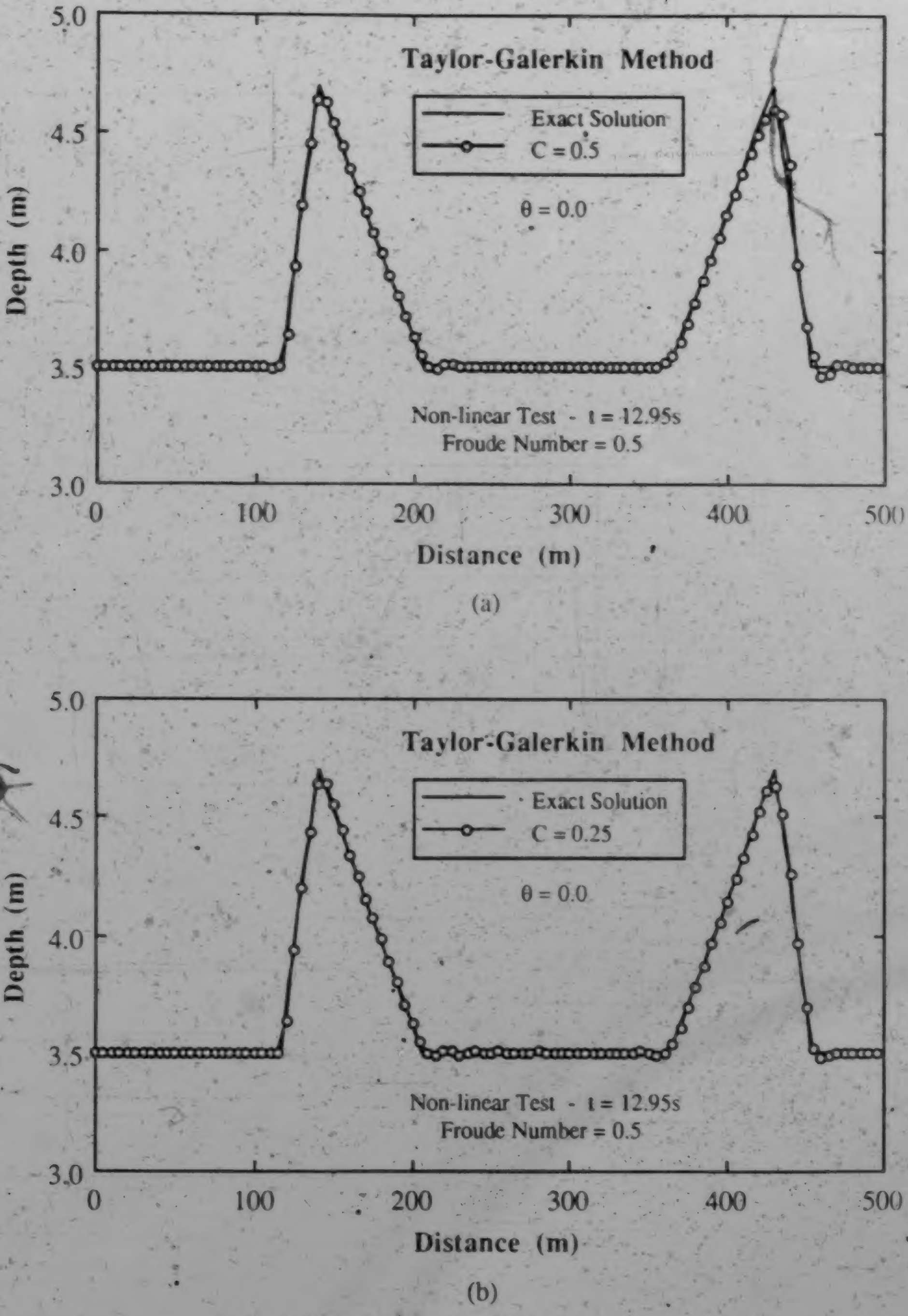
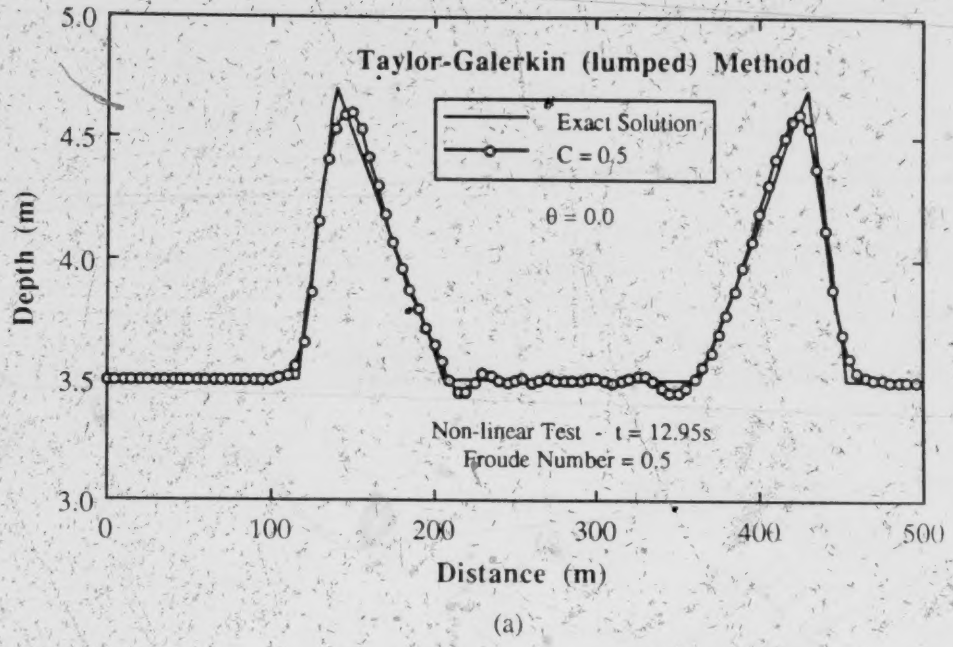
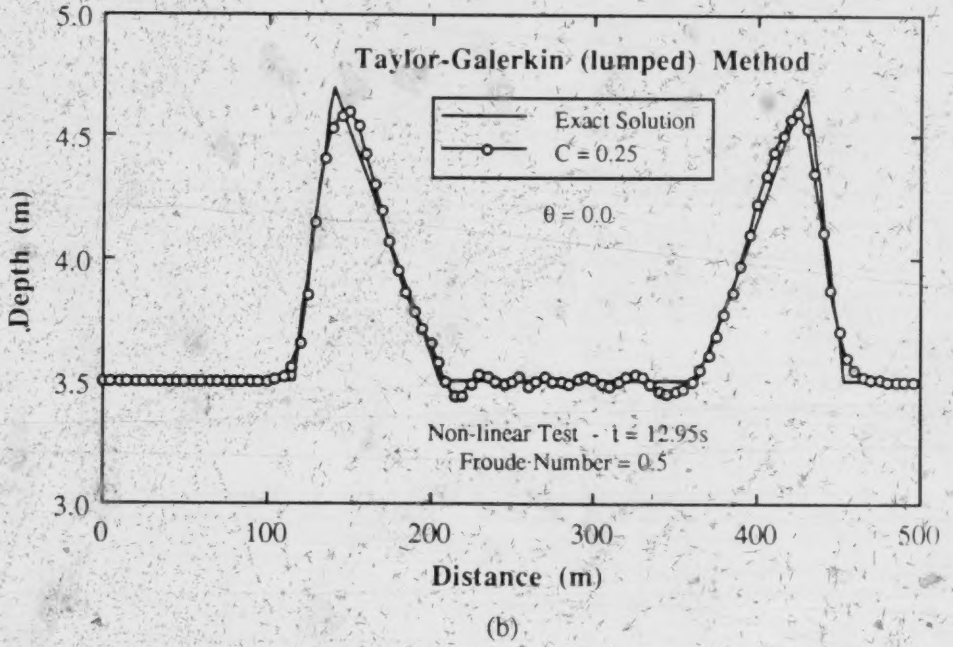


Figure 5.61 Non-linear perturbation (t = 12.95s), Taylor-Galerkin method, F = 0.5, θ = 0.0, (a) C = 0.5, Δt = 0.2159s; (b) C = 0.25, Δt = 0.1079s.

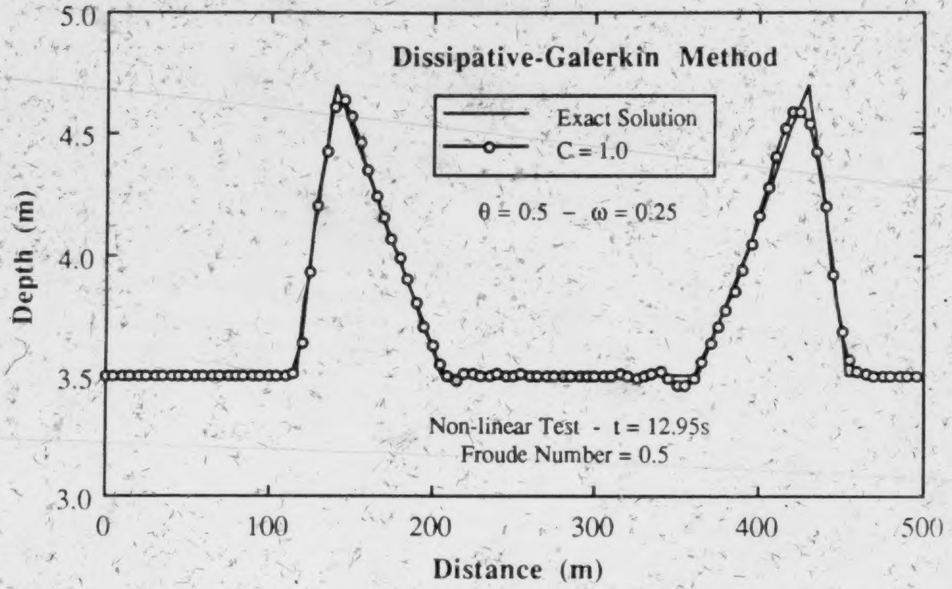


(a)

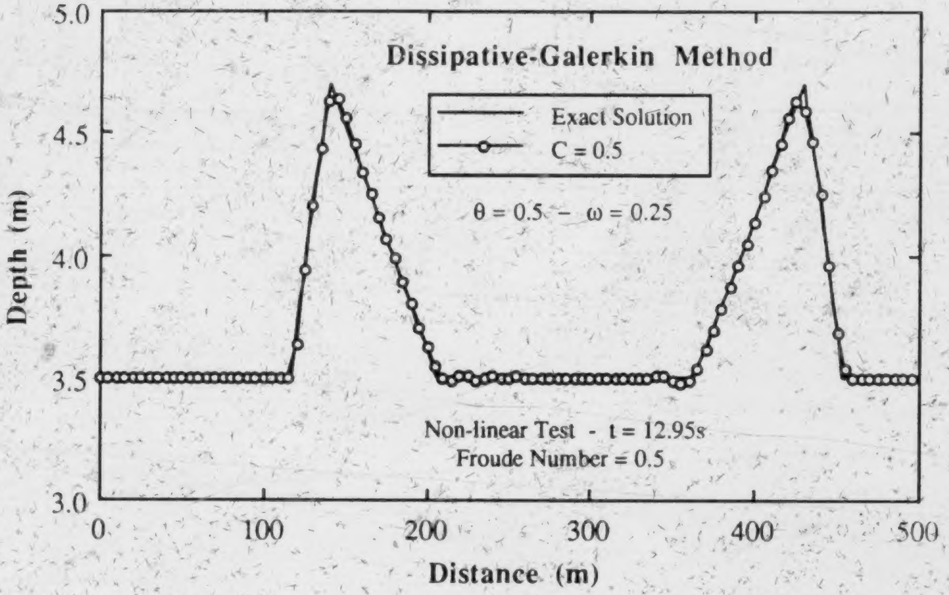


(b)

Figure 5.62 Non-linear perturbation (t = 12.95s), Taylor-Galerkin (lumped) method, **F** = 0.5, θ = 0.0 (a) C = 0.5, Δt = 0.2159s; (b) C = 0.25, Δt = 0.1079s.

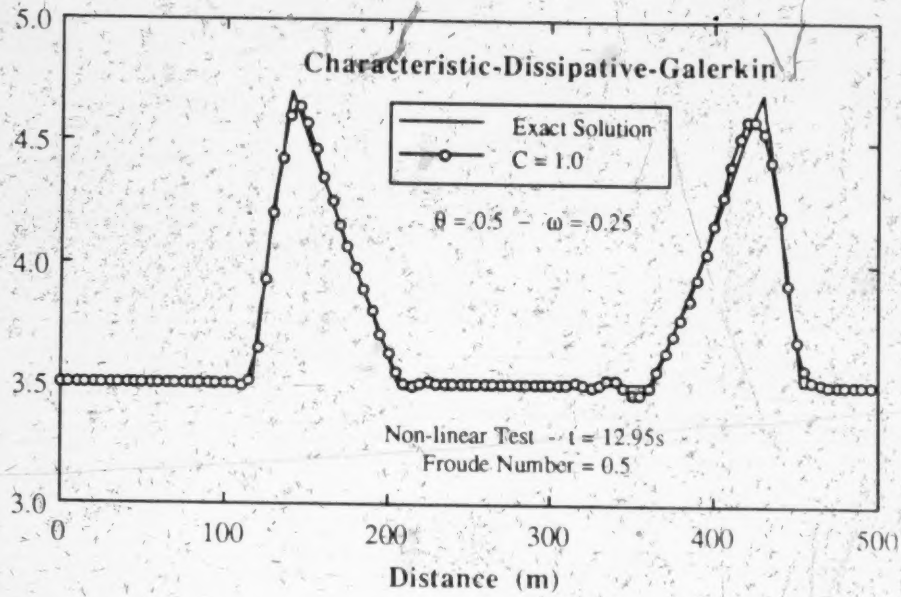


(a)

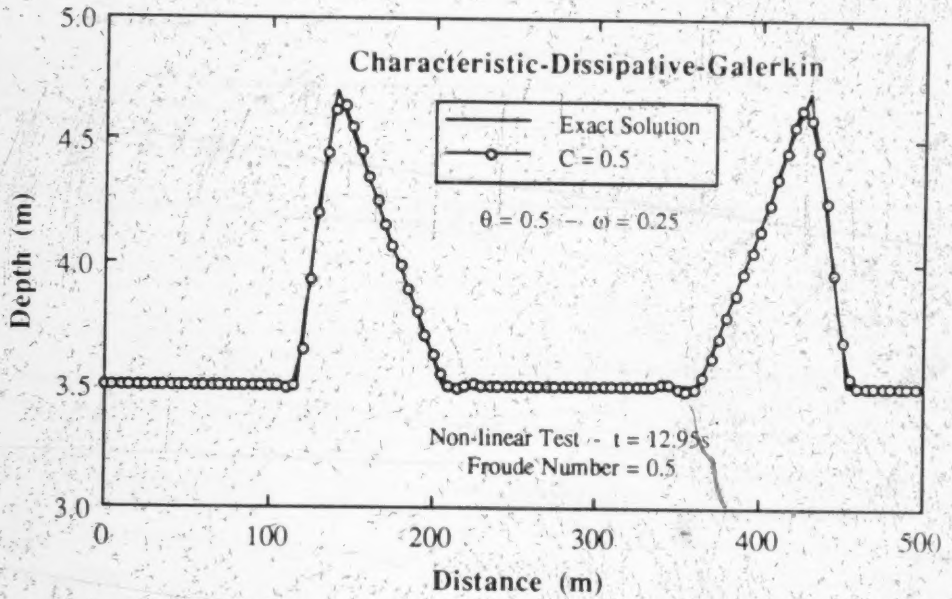


(b)

Figure 5.63 Non-linear perturbation (t = 12.95s), Dissipative-Galerkin method, **F** = 0.5, θ = 0.5, ω = 0.25, (a) C = 1.0, Δt = 0.4317s; (b) C = 0.5, Δt = 0.2159s.

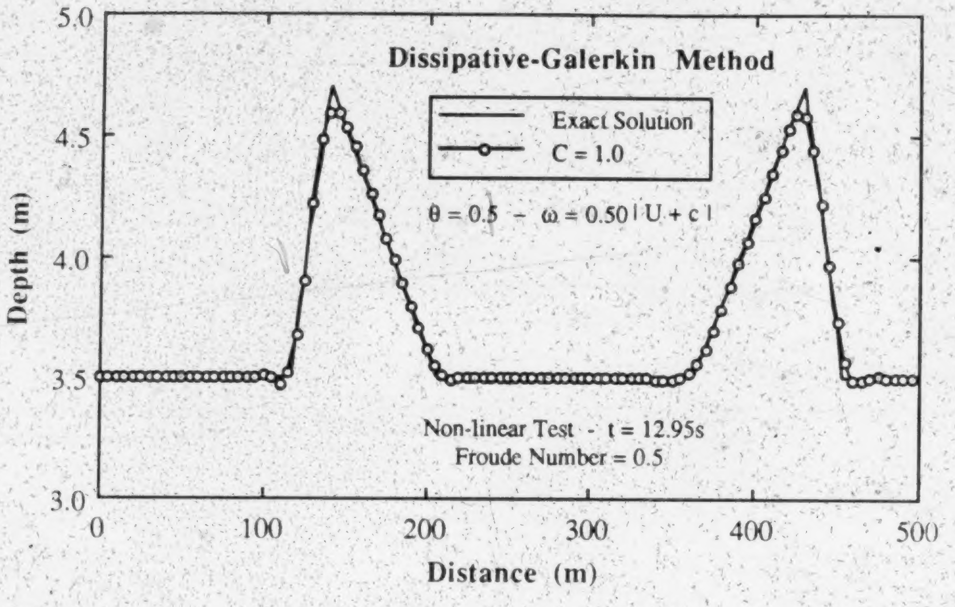


(a)

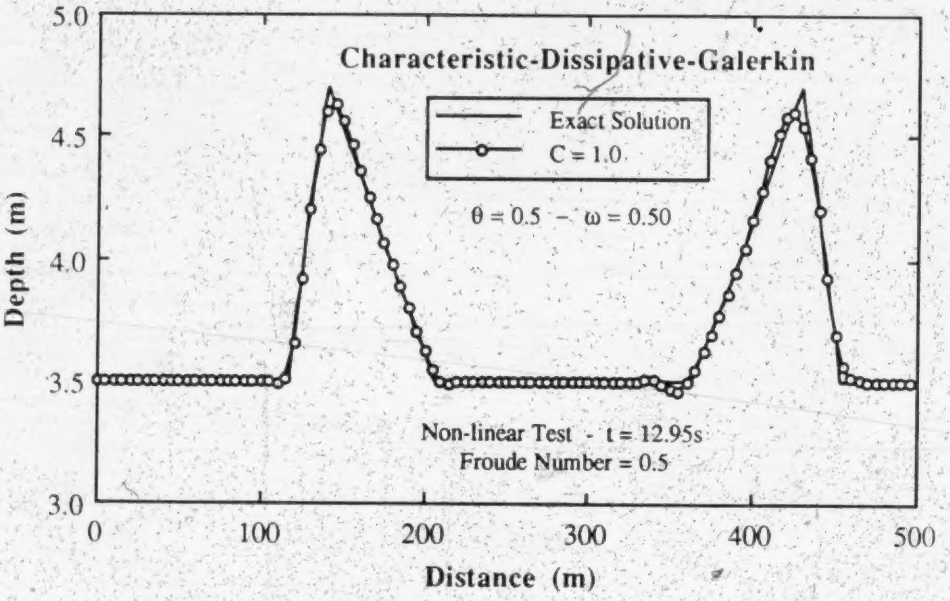


(b)

Figure 5.64 Non-linear perturbation (t = 12.95s), Characteristic-Dissipative-Gâlerkin method, **F** = 0.5, θ = 0.5, ω = 0.25, (a) C = 1.0, Δt = 0.4317s; (b) C = 0.5, Δt = 0.2159s.

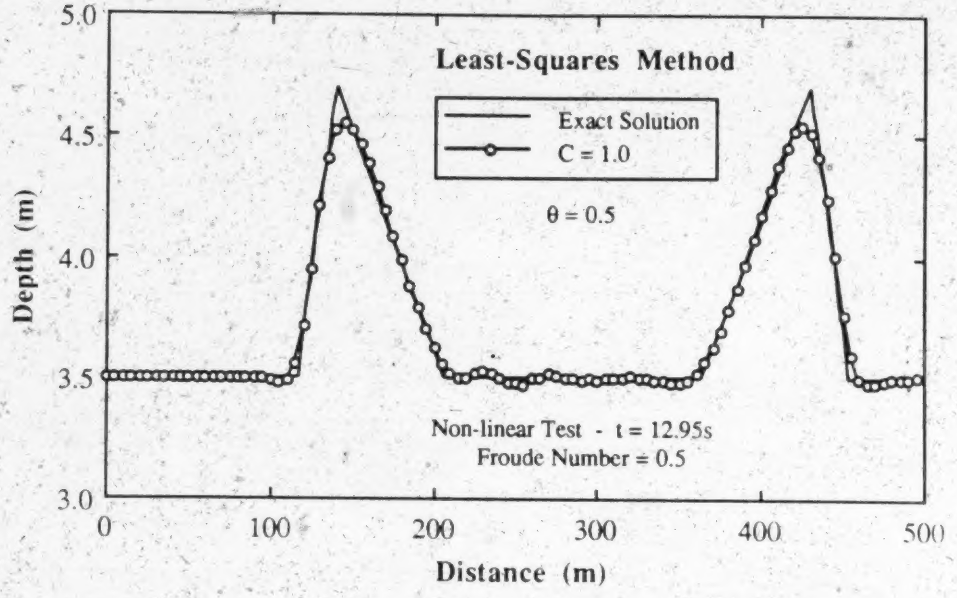


(a)

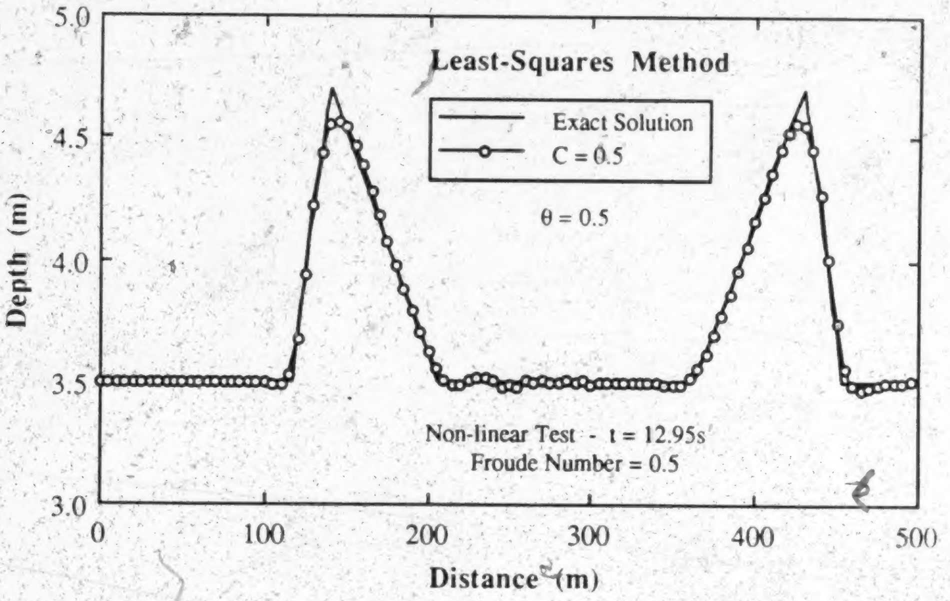


(b)

Figure 5.65 Non-linear perturbation (t = 12.95s), **F** = 0.5, θ = 0.5, C = 1.0, Δt = 0.4317s, (a) Dissipative-Galerkin method, ω = 0.50 | U + c |; (b) Characteristic-Dissipative-Galerkin method, ω = 0.50.

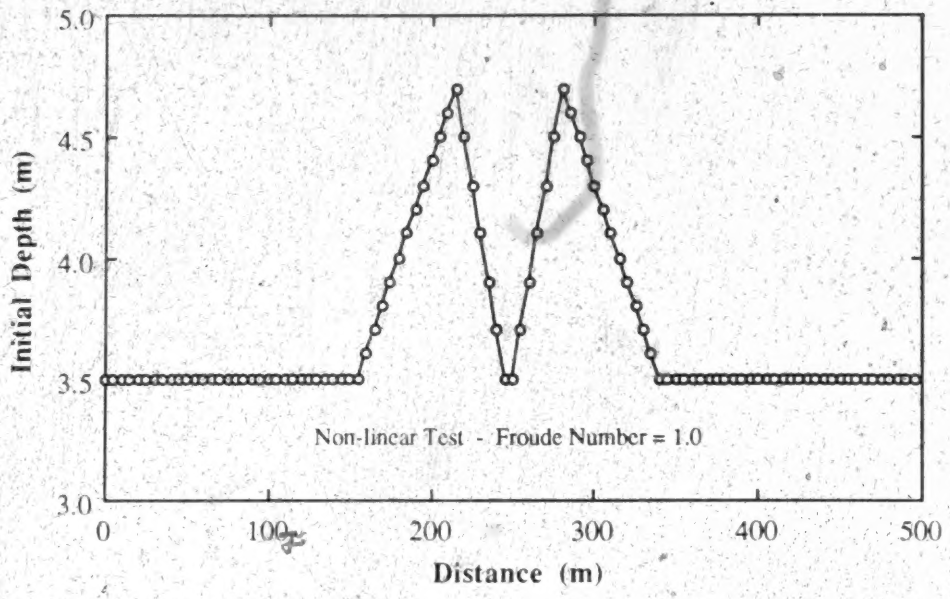


(a)



(b)

Figure 5.66 Non-linear perturbation ($t = 12.95s$), Least-Squares method, $\mathbf{F} = 0.5$, $\theta = 0.5$, (a) $C = 1.0$, $\Delta t = 0.4317s$; (b) $C = 0.5$, $\Delta t = 0.2159s$.



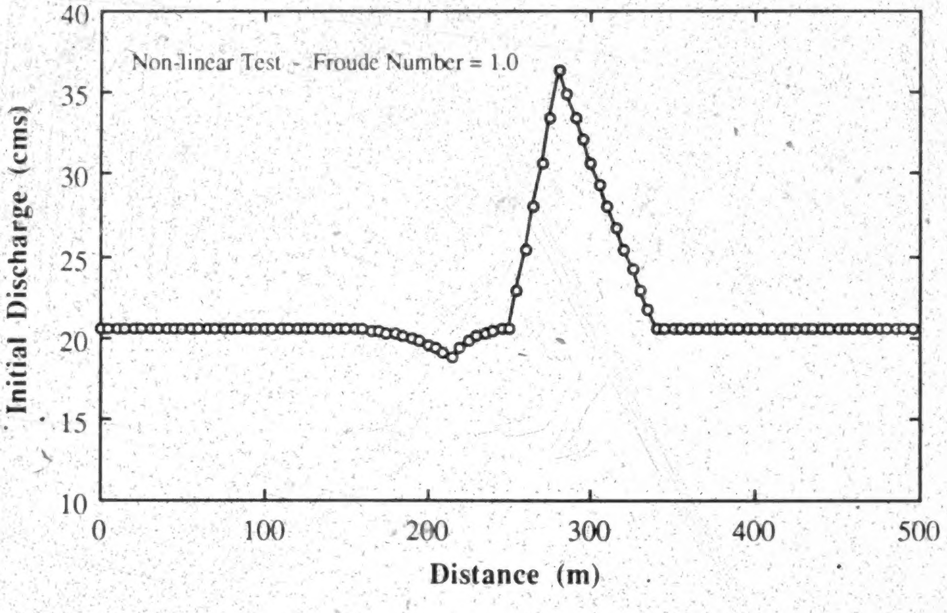
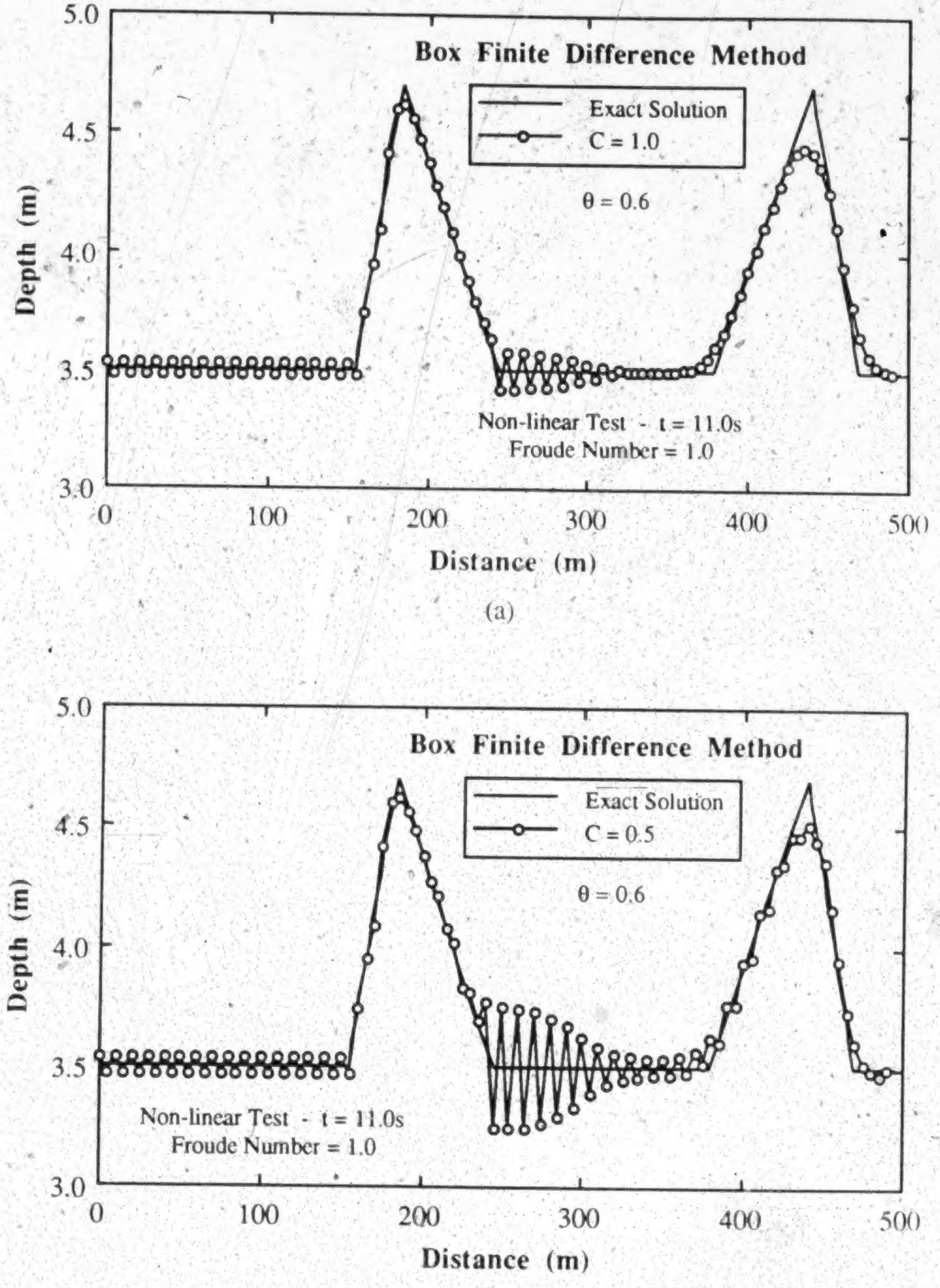


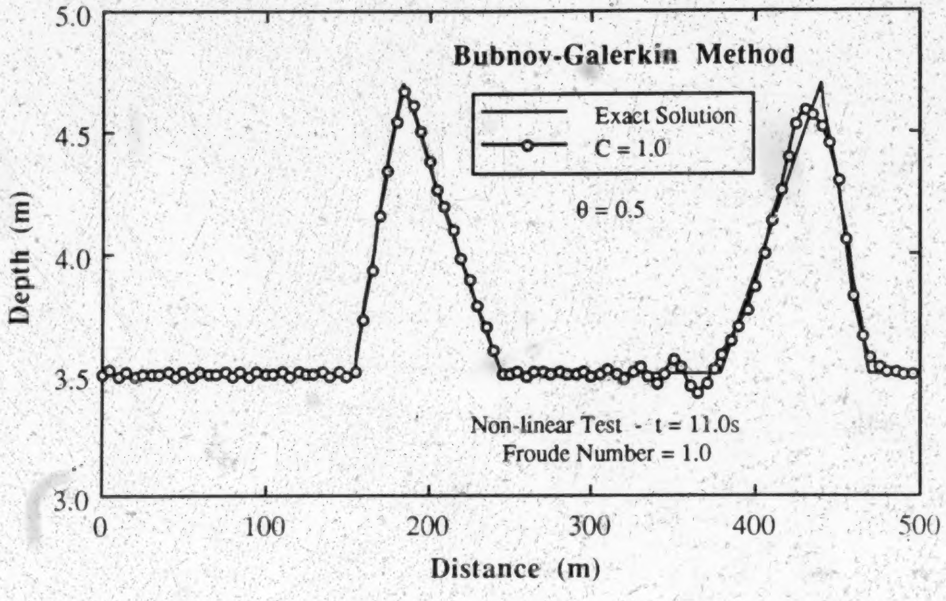
Figure 5.67 Initial conditions for non-linear test, **F** = 1.0.



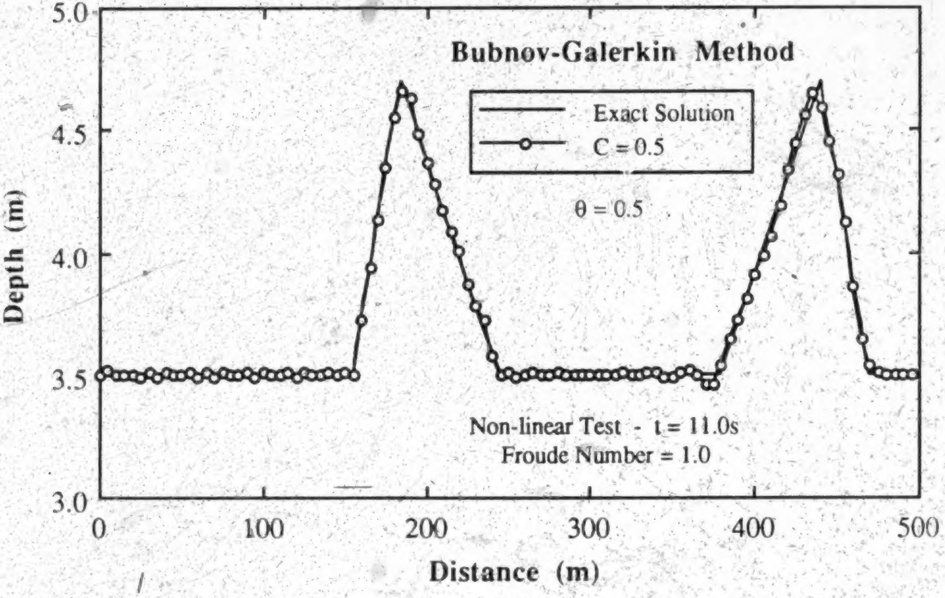
(a)

(b)

Figure 5.68 Non-linear perturbation (t = 11.0s), Box Finite Difference method, **F** = 1.0, θ = 0.5, (a) C = 1.0, Δt = 0.3670s; (b) C = 0.5, Δt = 0.1835s.

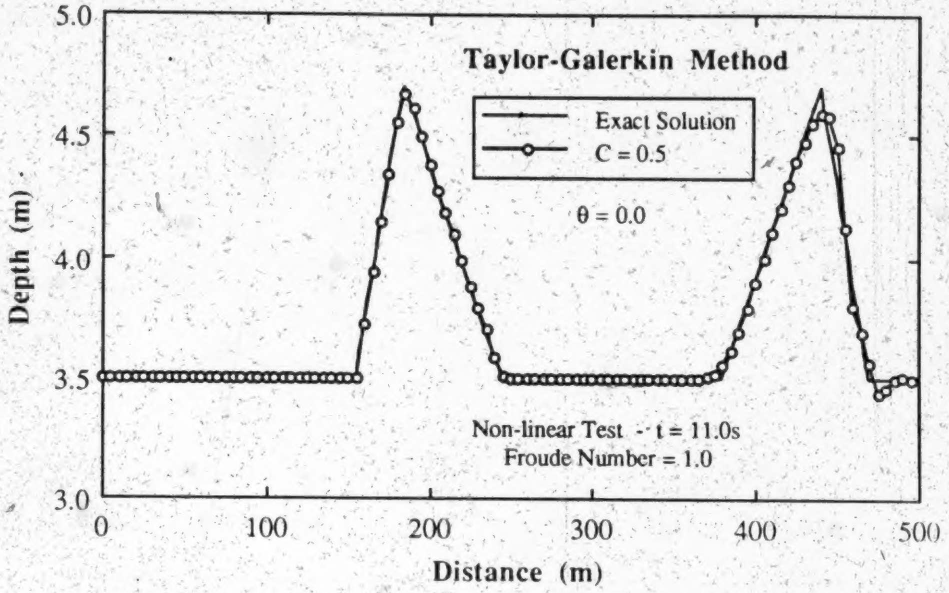


(a)

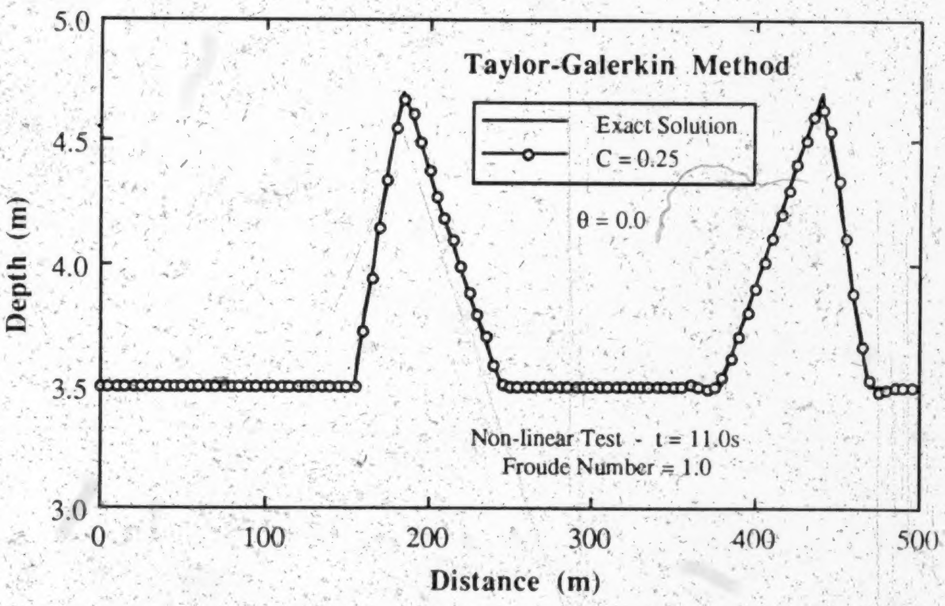


(b)

Figure 5.69 Non-linear perturbation (t = 11.0s), Bubnov-Galerkin method, **F** = 1.0, θ = 0.5 (a) C = 1.0, Δt = 0.3670s; (b) C = 0.5, Δt = 0.1835s.

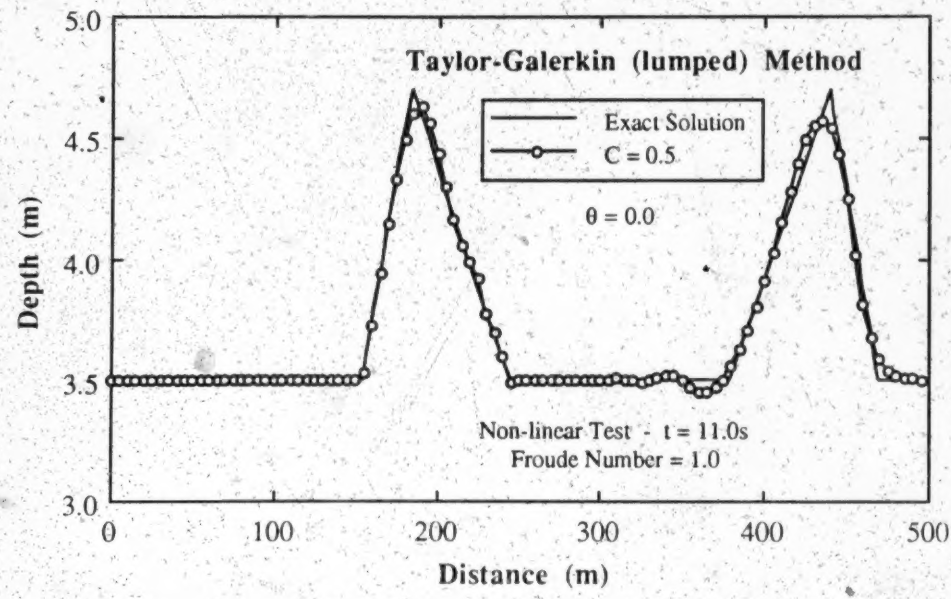


(a)

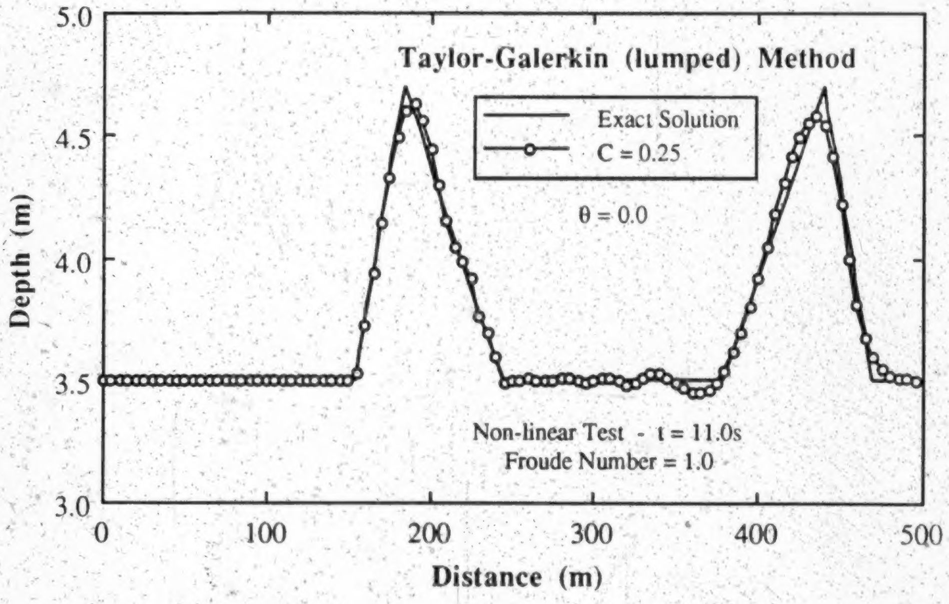


(b)

Figure 5.70 Non-linear perturbation (t = 11.0s), Taylor-Galerkin method, **F** = 1.0, θ = 0.0, (a) C = 0.5, Δt = 0.1835s; (b) C = 0.25, Δt = 0.09175s.



(a)



(b)

Figure 5.71 Non-linear perturbation (t = 11.0s), Taylor-Galerkin (lumped) method, F = 0.0, θ = 0.5 (a) C = 0.5, Δt = 0.1835s; (b) C = 0.25, Δt = 0.09175s.

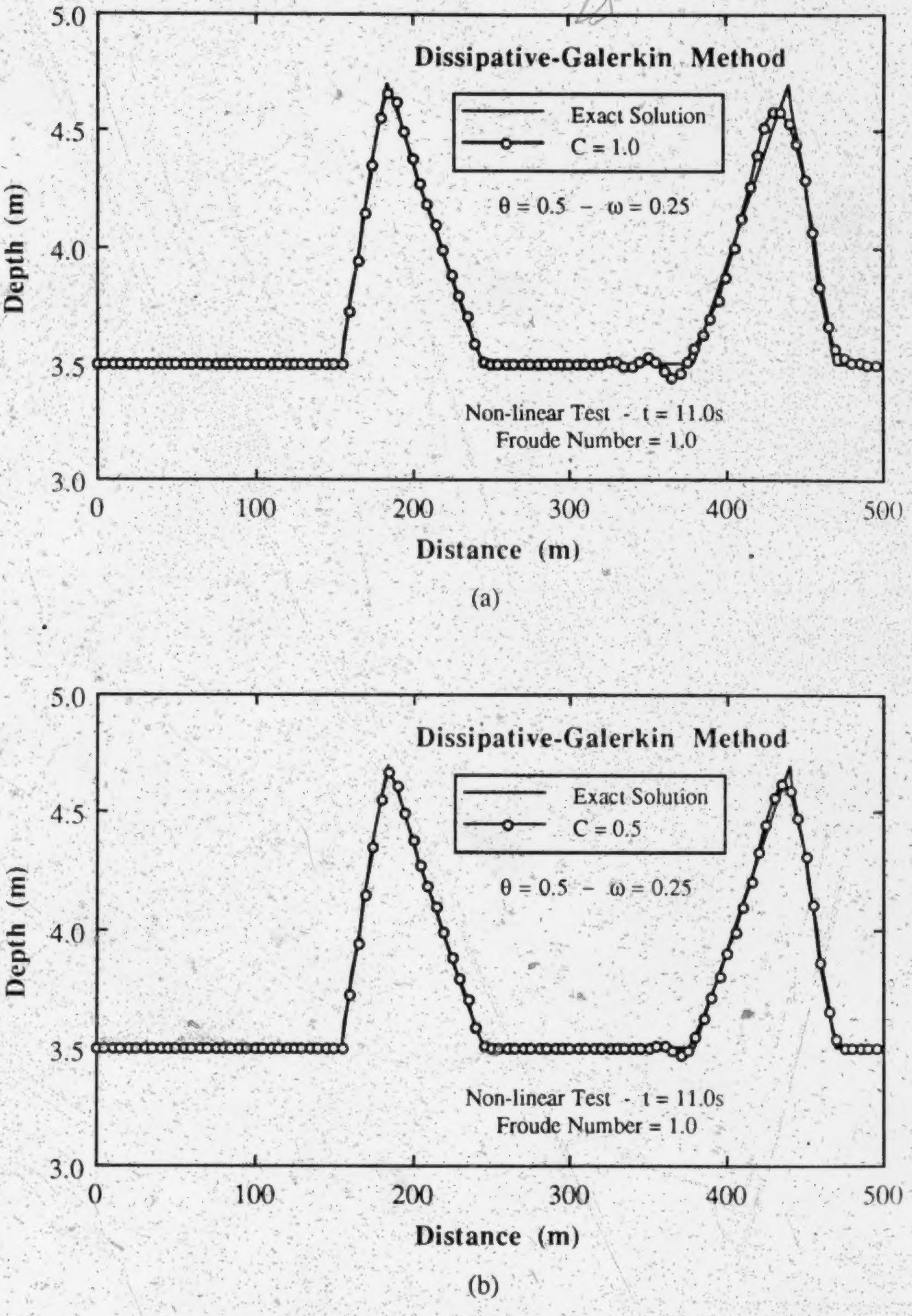
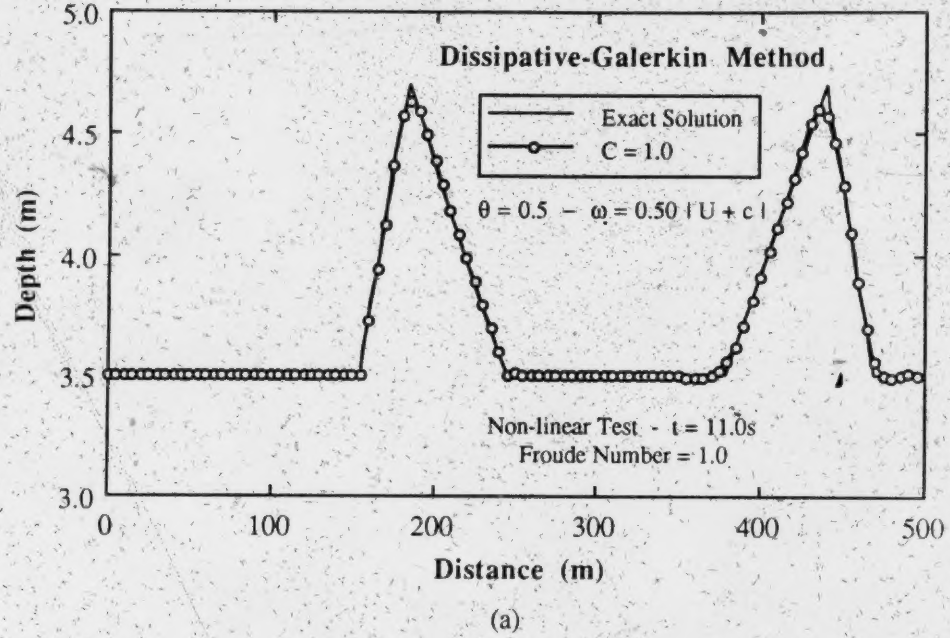


Figure 5.72 Non-linear perturbation (t = 11.0s), Dissipative-Galerkin method, **F** = 1.0, θ = 0.5, ω = 0.25, (a) C = 1.0, Δt = 0.3670s; (b) C = 0.5, Δt = 0.1835s.



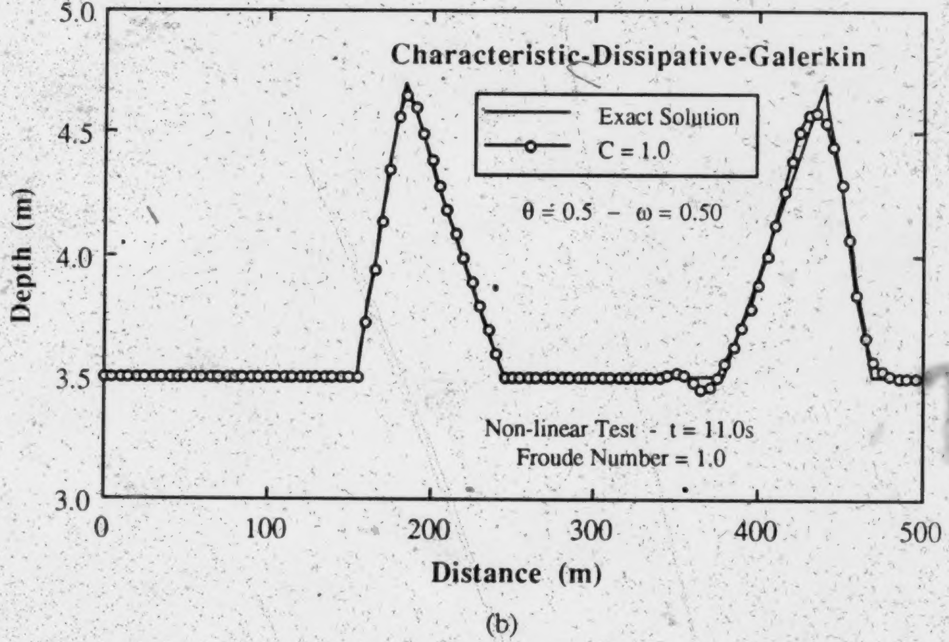
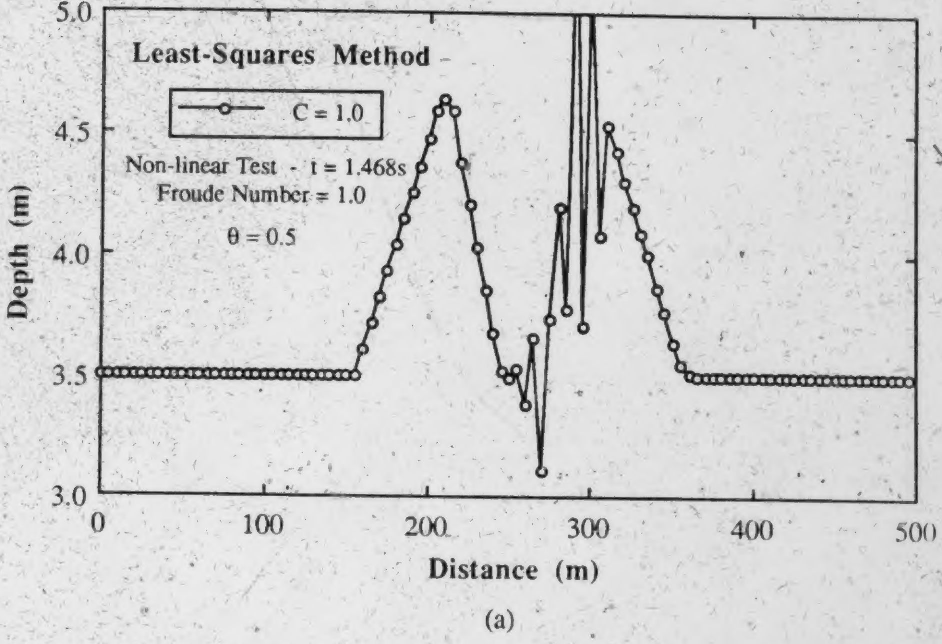


Figure 5.73 Non-linear perturbation (t = 11.0s), **F** = 1.0, θ = 0.5, C = 1.0, Δt = 0.3670s, (a) Dissipative-Galerkin method, ω = 0.50 | U + c |; (b) Characteristic-Dissipative-Galerkin method, ω = 0.50.



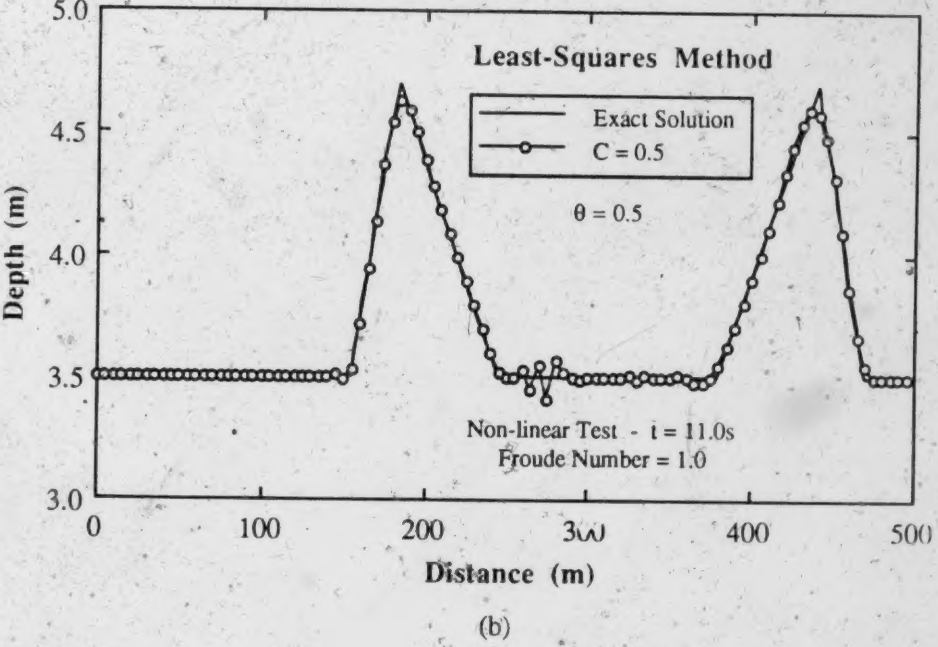
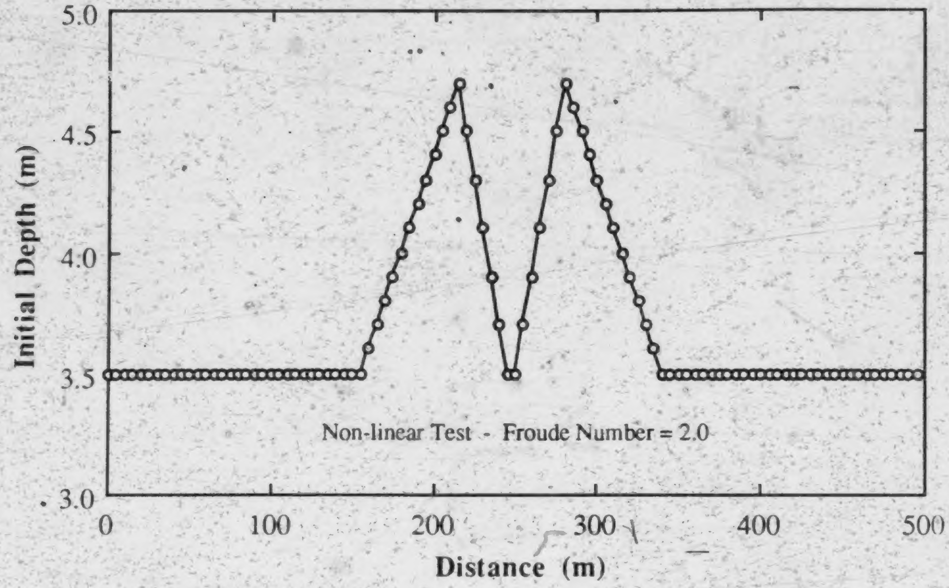


Figure 5.74 Non-linear perturbation, Least-Squares method, **F** = 1.0, θ = 0.5, (a) C = 1.0, Δt = 0.3670s, t = 1.468s; (b) C = 0.5, Δt = 0.1835s, t = 11.0s.



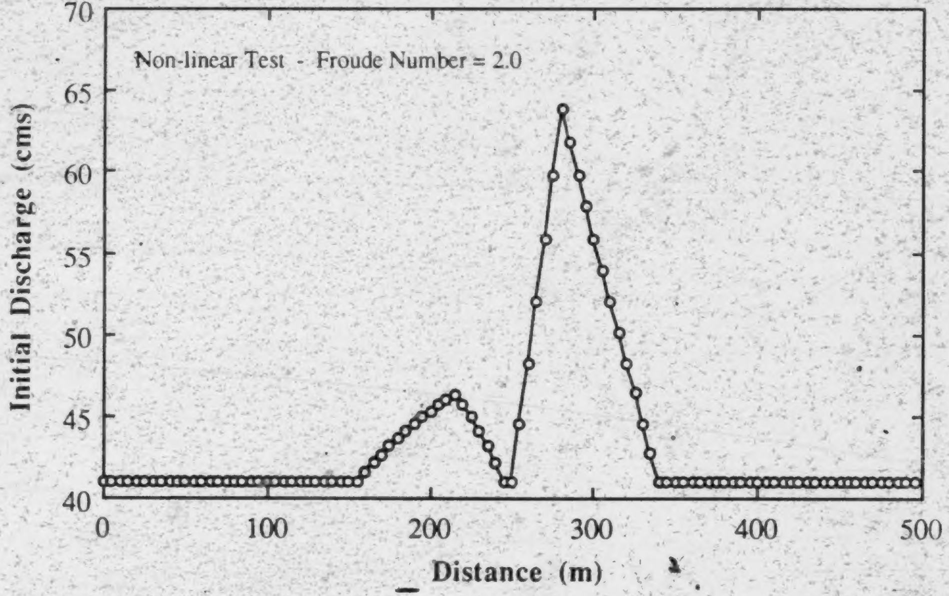


Figure 5.75 Initial conditions for non-linear test, **F** = 2.0.

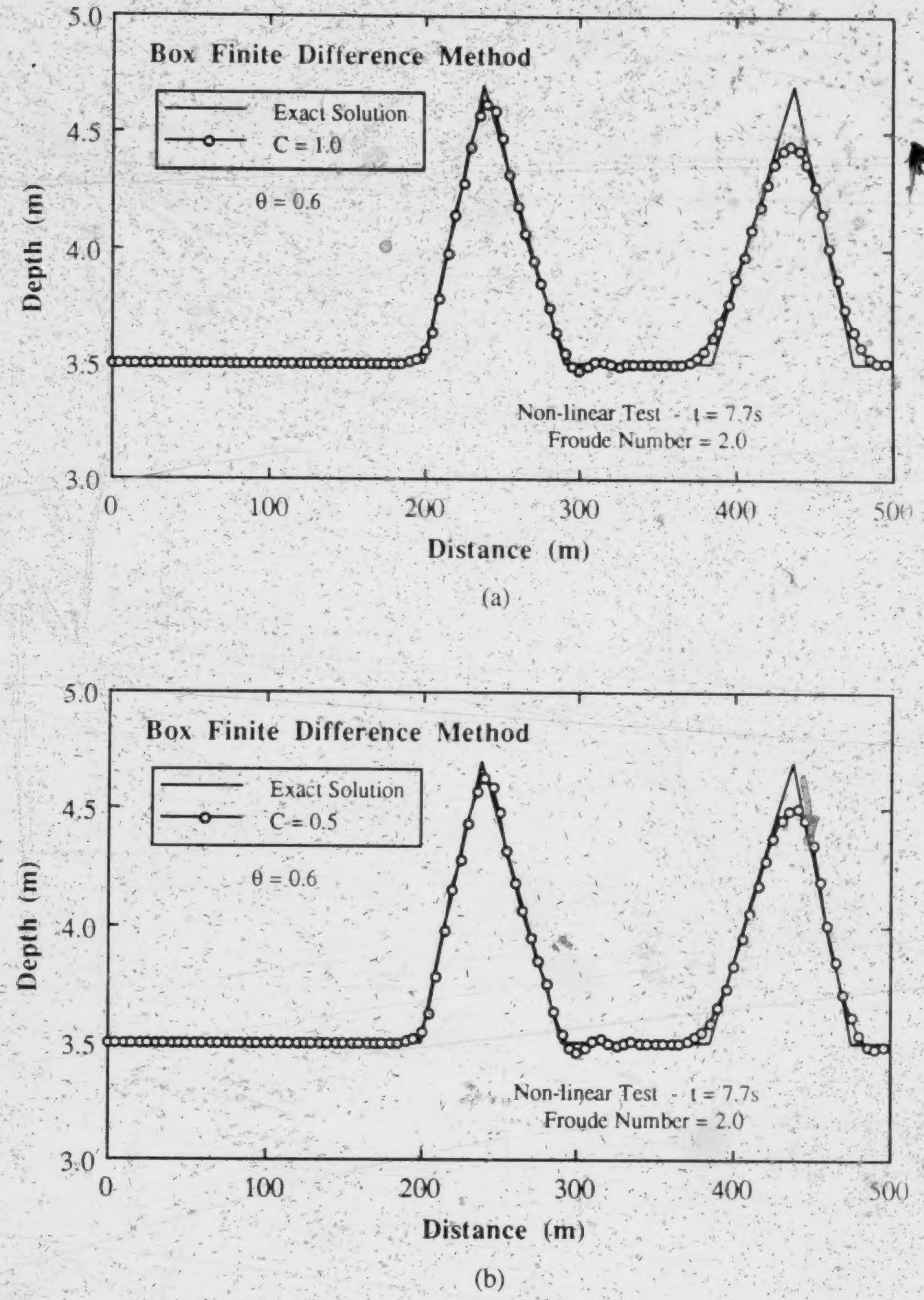
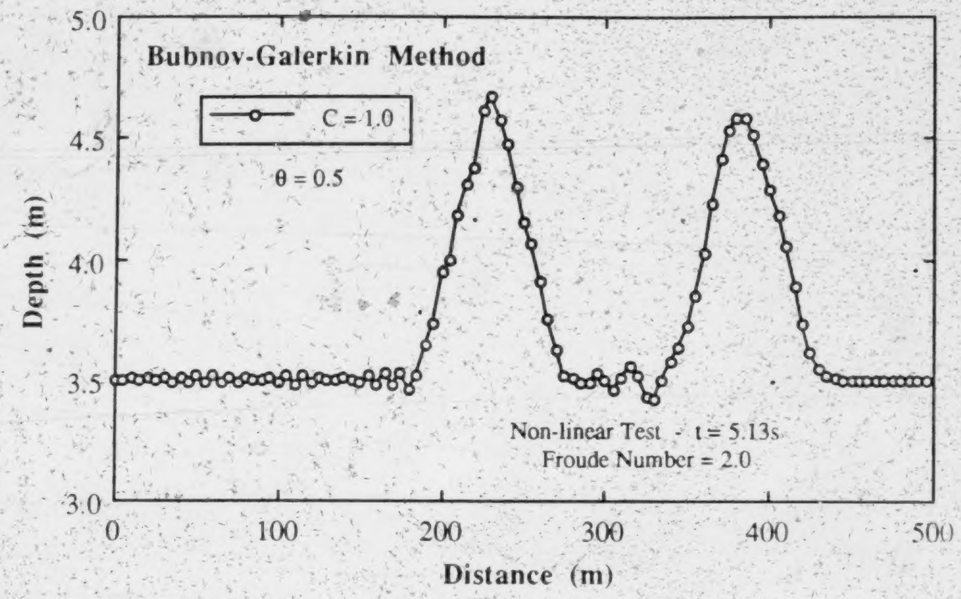
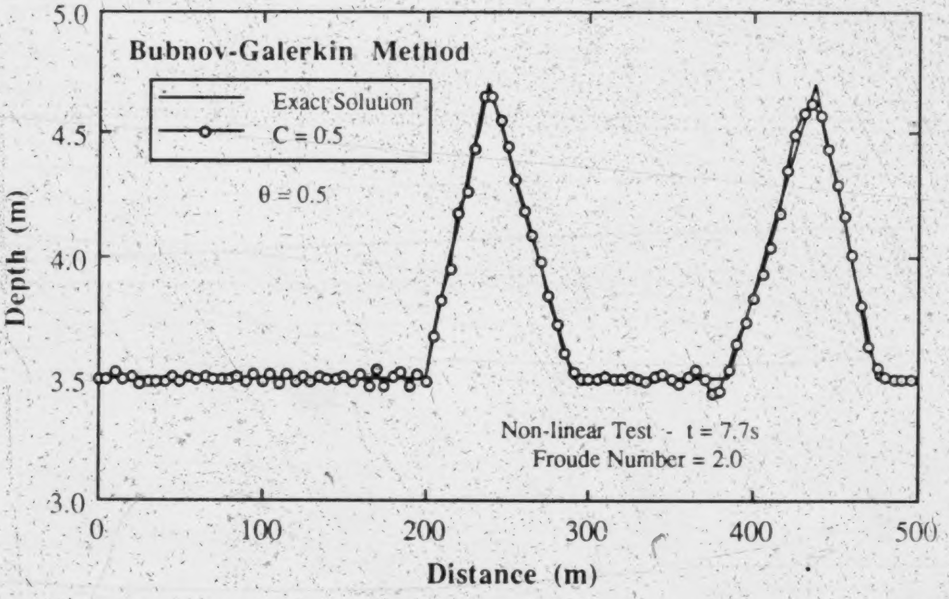


Figure 5.76 Non-linear perturbation (t = 7.7s), Box Finite Difference method, F = 2.0, θ = 0.5, (a) C = 1.0, Δt = 0.2566s; (b) C = 0.5, Δt = 0.1283s.

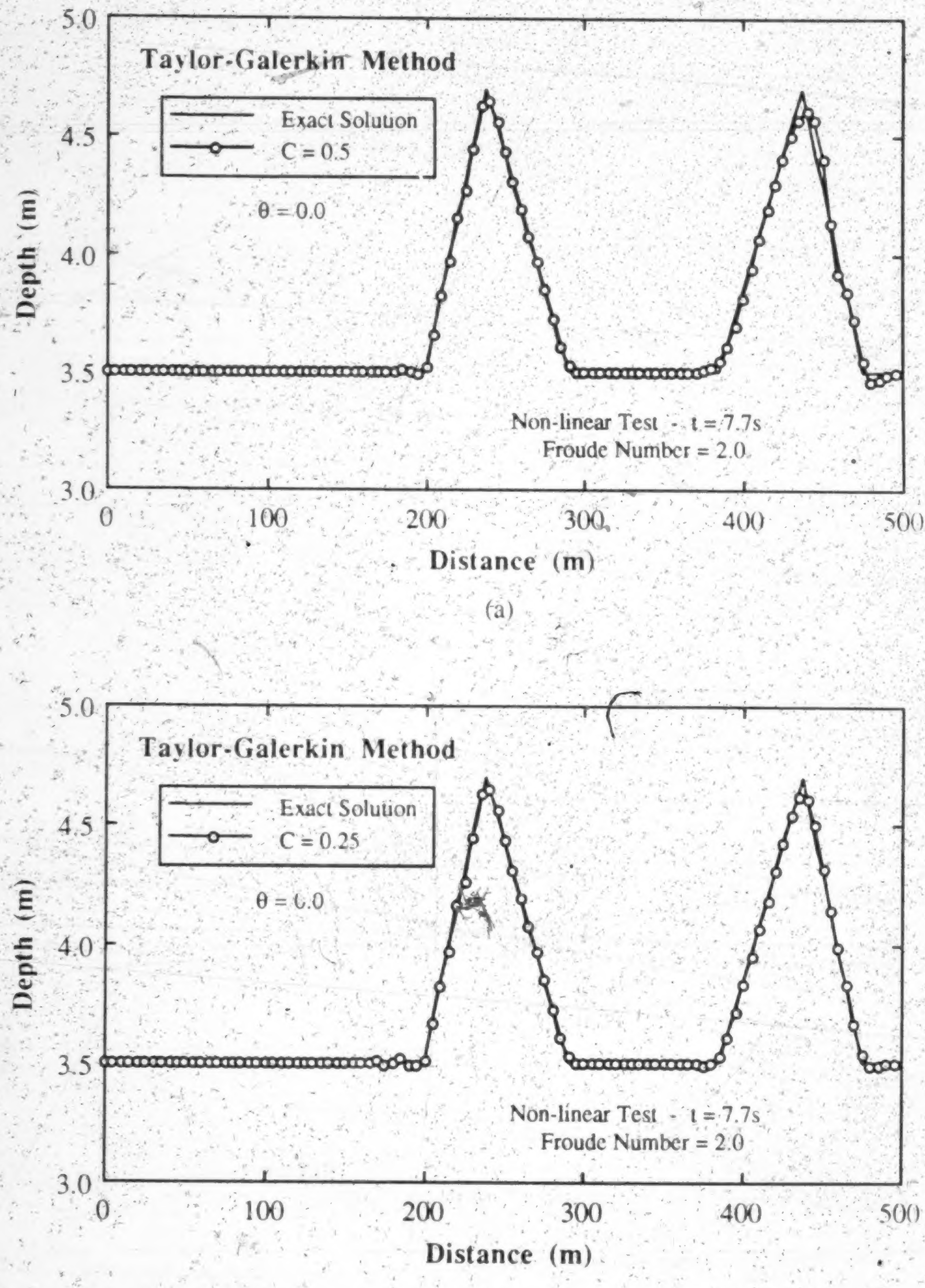


(a)



(b)

Figure 5.77 Non-linear perturbation, Bubnov-Galerkin method, F = 2.0, $\theta = 0.5$ (a) C = 1.0, t = 5.13s, $\Delta t = 0.2566$s; (b) C = 0.5, t = 7.7s, $\Delta t = 0.1283$s.



(a)

(b)

Figure 5.78 Non-linear perturbation (t = 7.7s), Taylor-Galerkin method, **F** = 2.0, θ = 0.0, (a) C = 0.5, Δt = 0.1283s; (b) C = 0.25, Δt = 0.06415s.

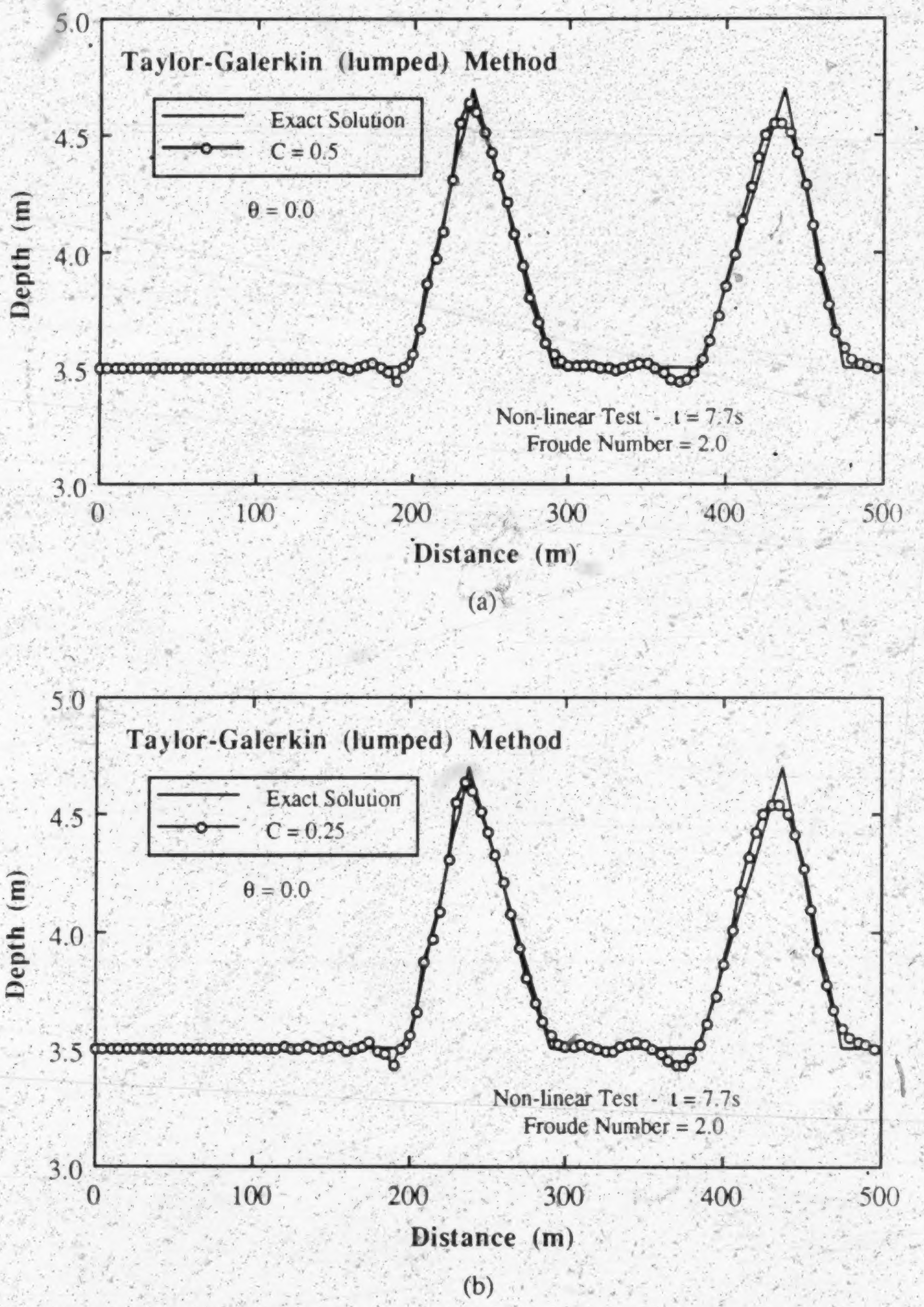
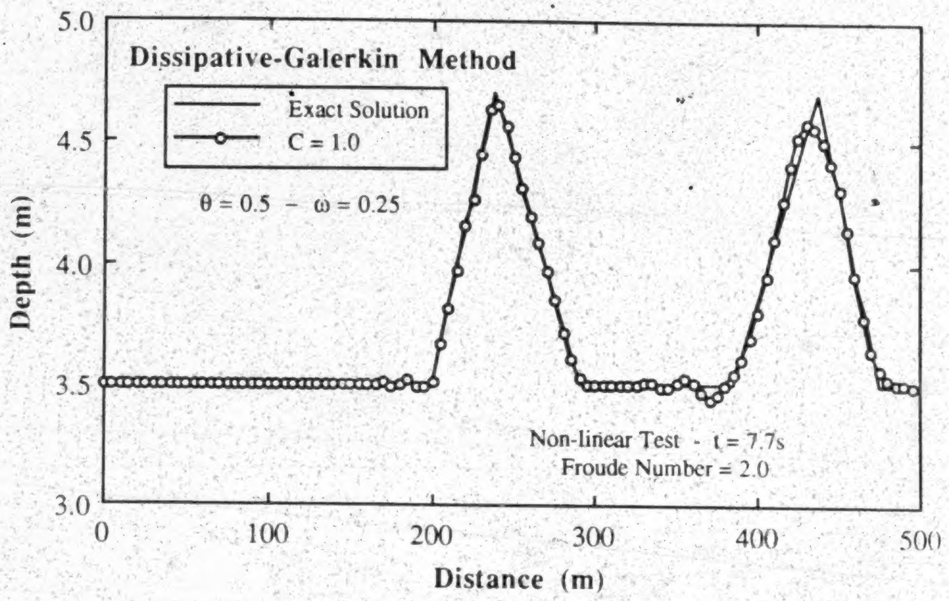
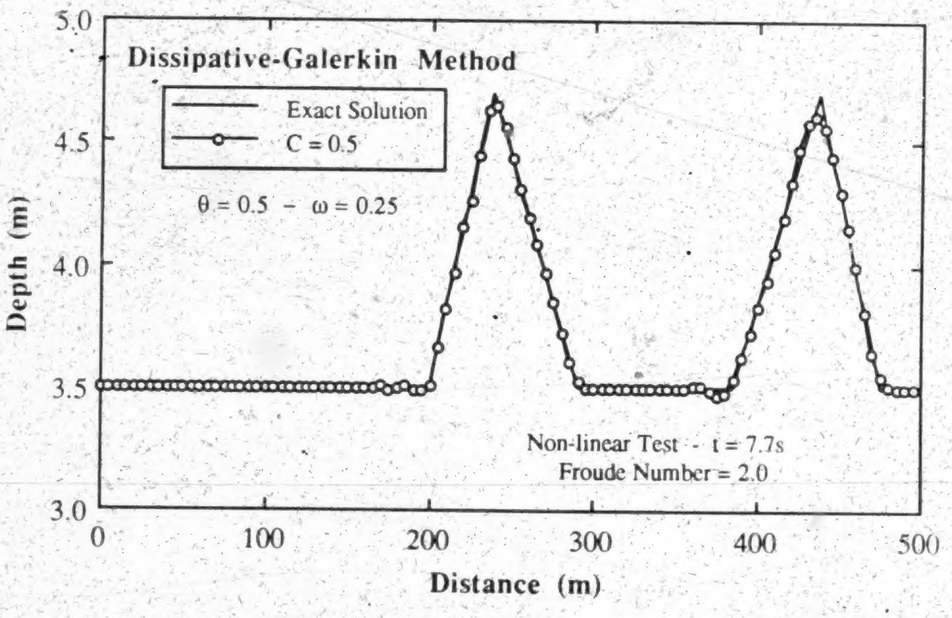


Figure 5.79 Non-linear perturbation (t = 7.7s), Taylor-Galerkin (lumped) method, **F** = 2.0, θ = 0.5 (a) C = 0.5, Δt = 0.1283s; (b) C = 0.25, Δt = 0.06415s.

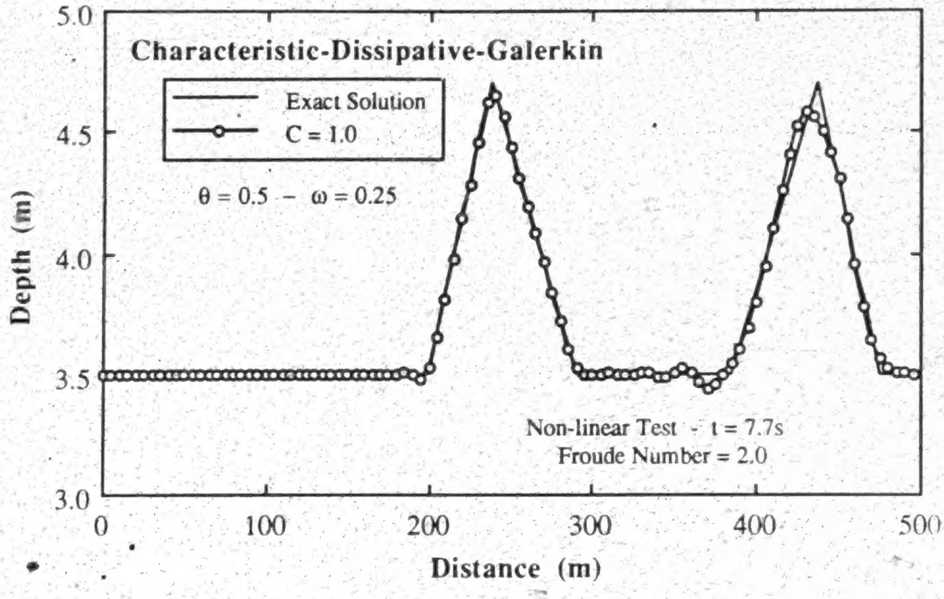


(a)

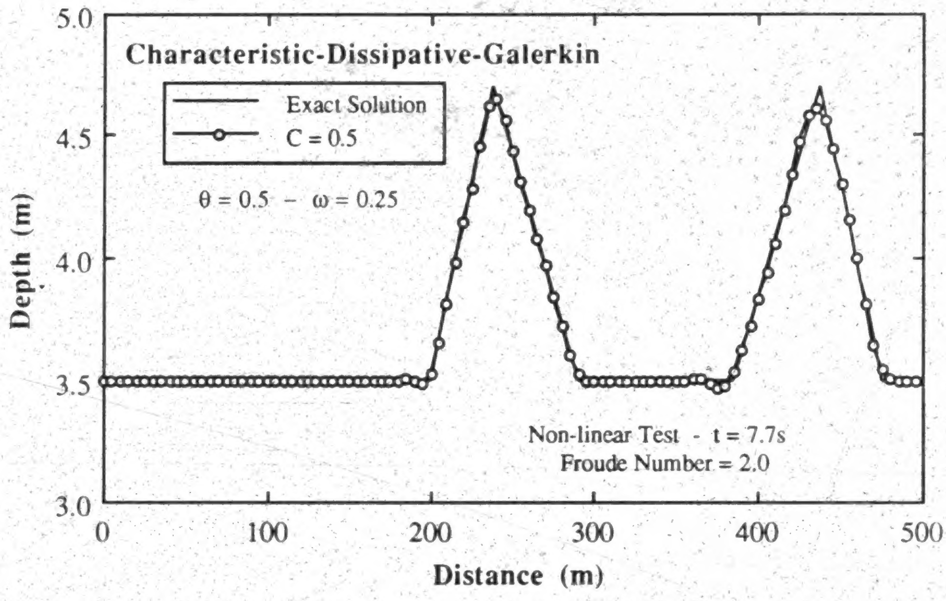


(b)

Figure 5.80 Non-linear perturbation (t = 7.7s), Dissipative-Galerkin method, **F** = 2.0, θ = 0.5, ω = 0.25, (a) C = 1.0, Δt = 0.2566s; (b) C = 0.5, Δt = 0.1283s.

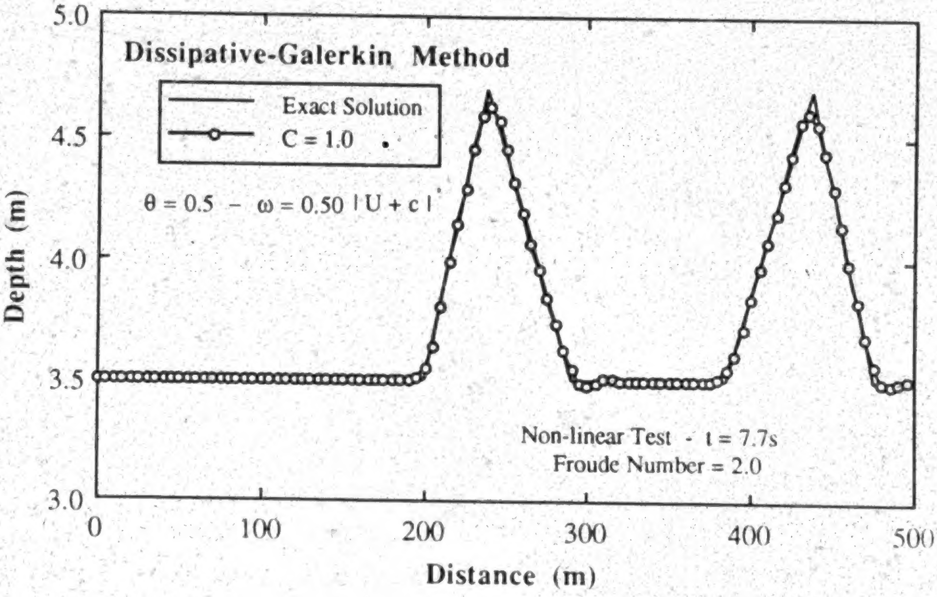


(a)

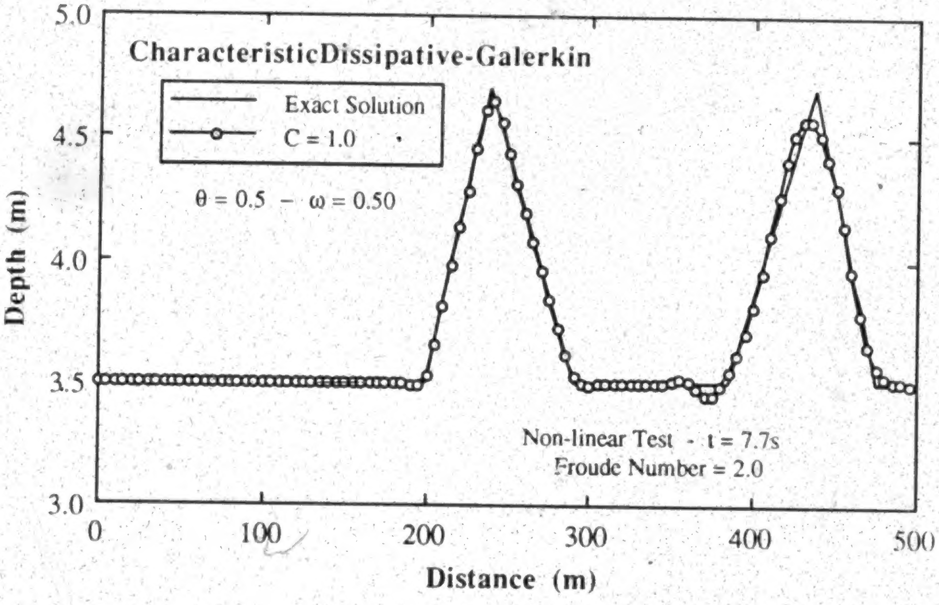


(b)

Figure 5.81 Non-linear perturbation (t = 7.7s), Characteristic-Dissipative-Galerkin method, **F** = 2.0, θ = 0.5, ω = 0.25, (a) C = 1.0, Δt = 0.2566s; (b) C = 0.5, Δt = 0.1283s.

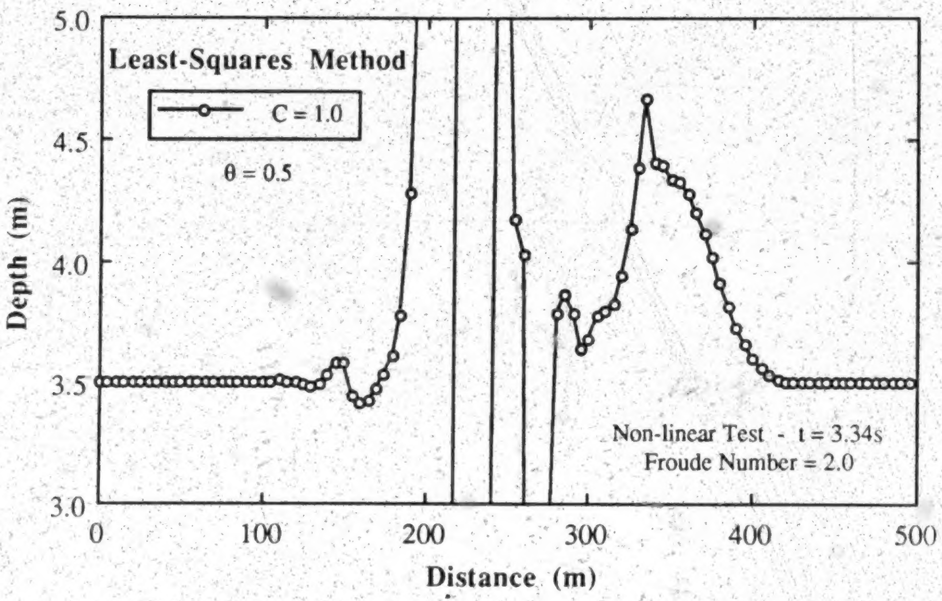


(a)

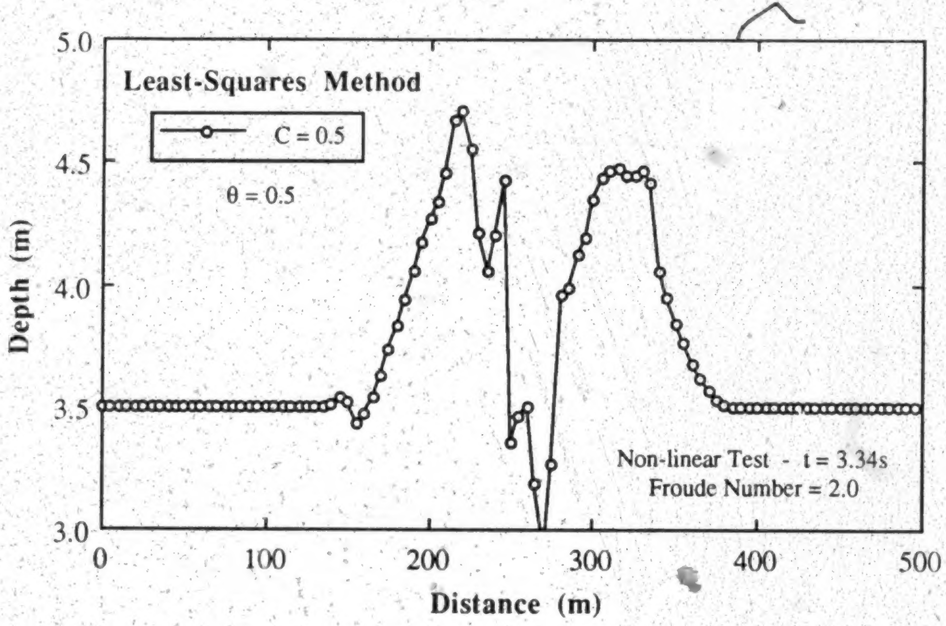


(b)

Figure 5.82 Non-linear perturbation (t = 7.7s), F = 2.0, θ = 0.5, C = 1.0, Δt = 0.2566s, (a) Dissipative-Galerkin method, ω = 0.50 | U + c |; (b) Characteristic-Dissipative-Galerkin method, ω = 0.50.



(a)



(b)

Figure 5.83 Non-linear perturbation (t = 3.34s), Least-Squares method, **F** = 2.0, θ = 0.5, (a) C = 1.0, Δt = 0.2566s; (b) C = 0.5, Δt = 0.1283s.

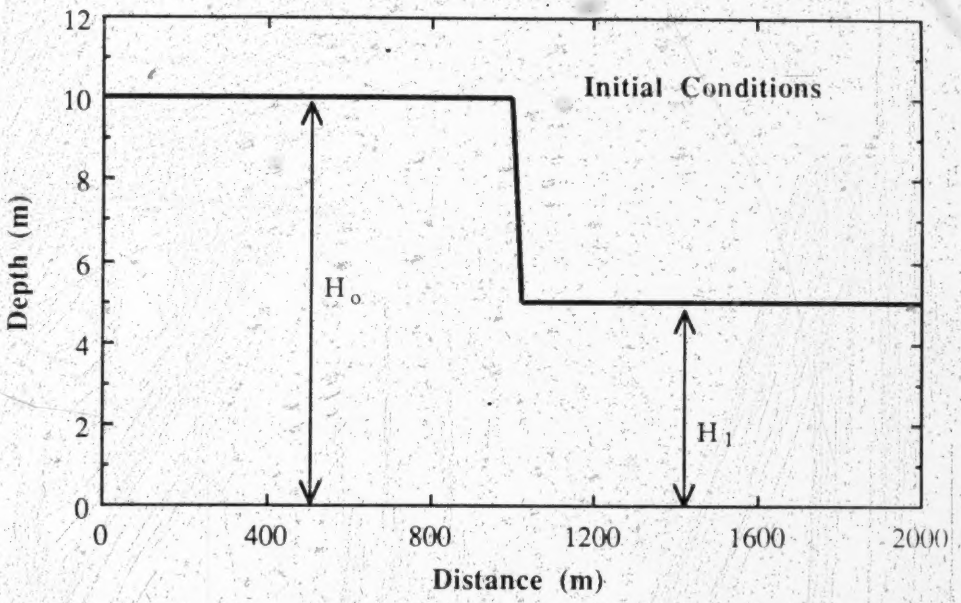
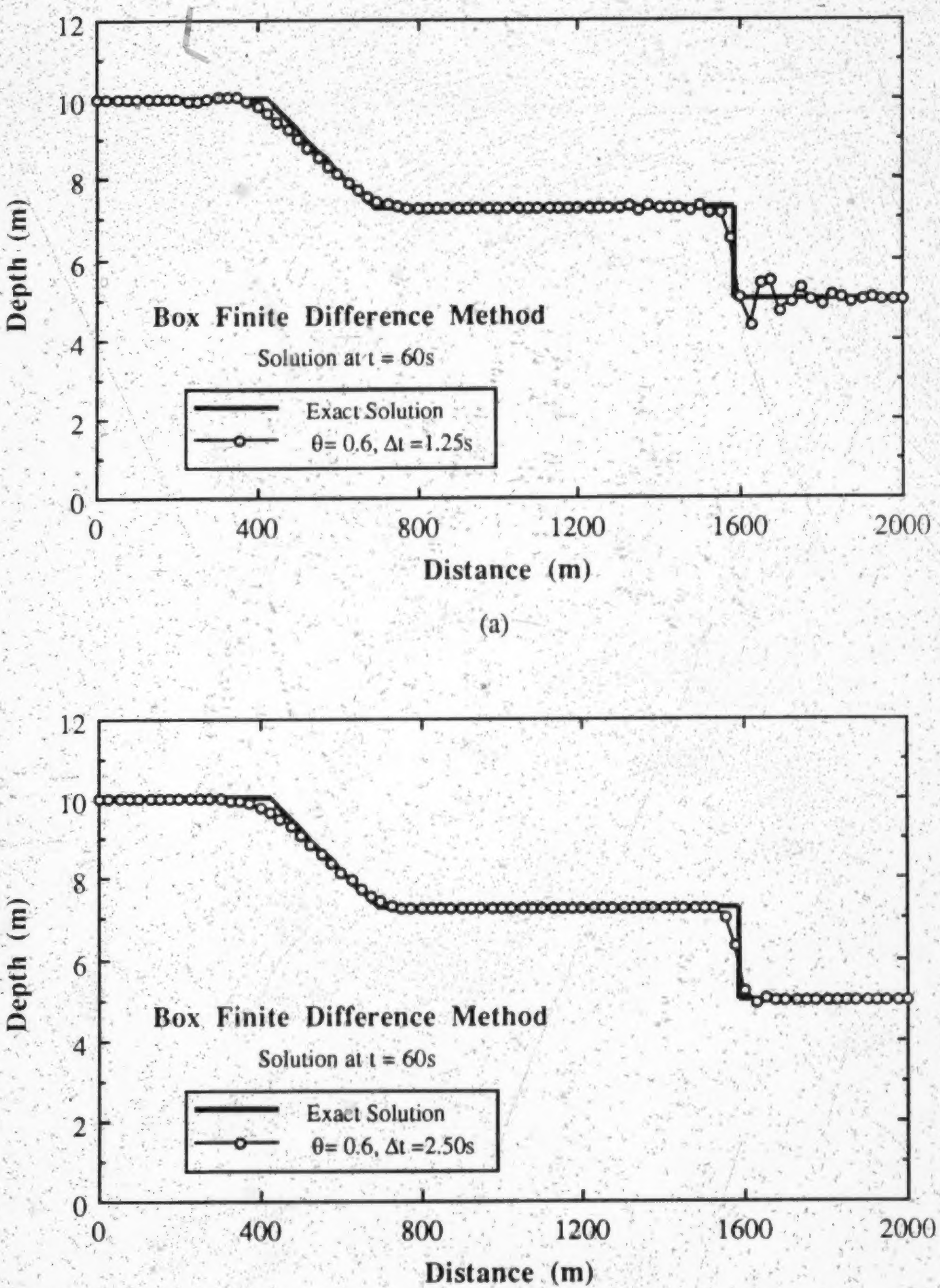


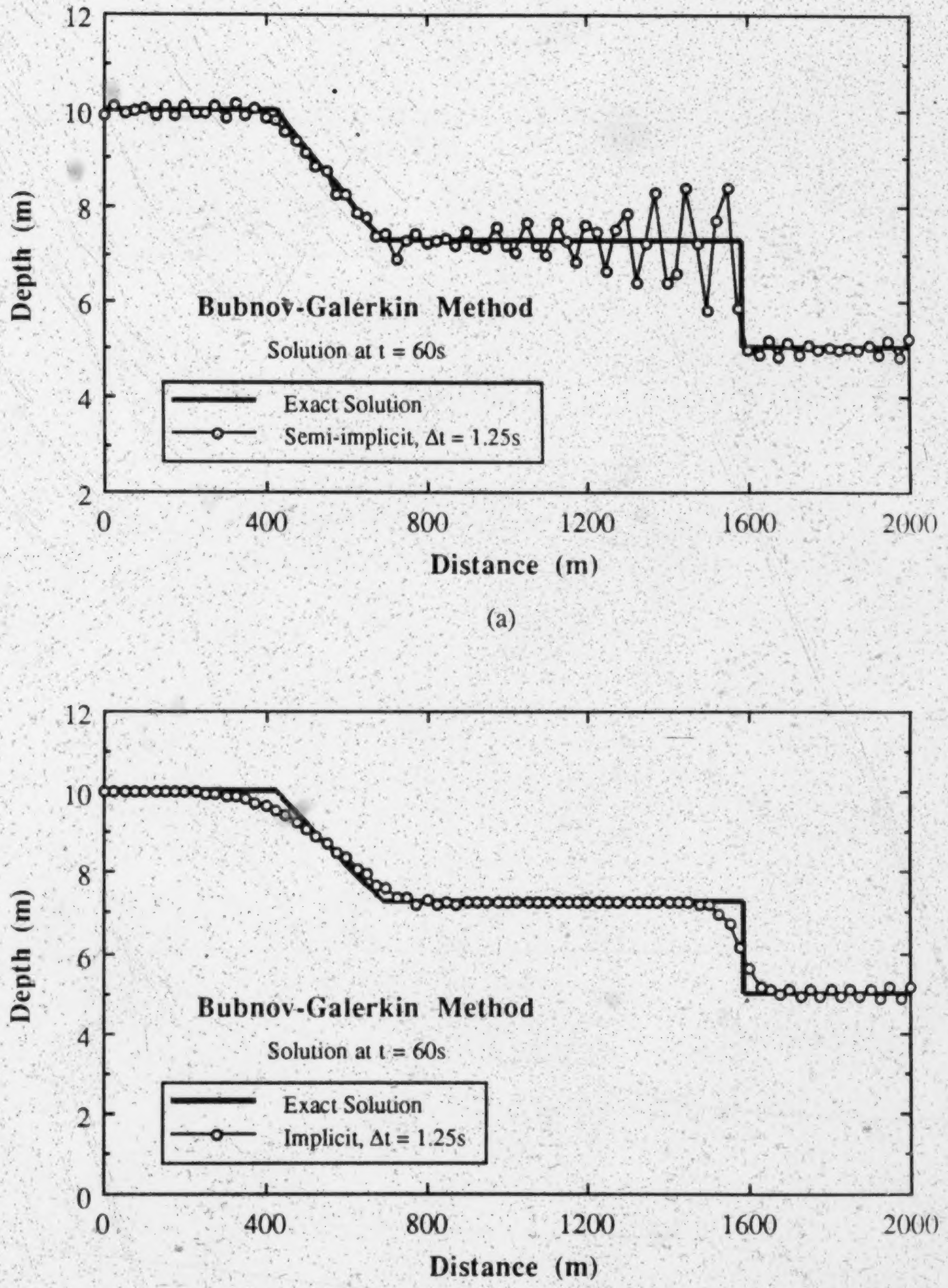
Figure 5.84 Dambreak, $H_1/H_o = 0.5$, (t = 0s), initial conditions.



(a)

(b)

Figure 5.85 Dambreak, $H_1/H_o = 0.5$, ($t = 60$s), Box Finite Difference method, $\theta = 0.6$: (a) $\Delta t = 1.25$s; (b) $\Delta t = 2.50$s.



(a)

(b)

Figure 5.86 Dambreak, $H_f/H_o = 0.5$, (t = 60s), Bubnov-Galerkin method, $\Delta t = 1.25$s: (a) $\theta = 0.5$; (b) $\theta = 1.0$.

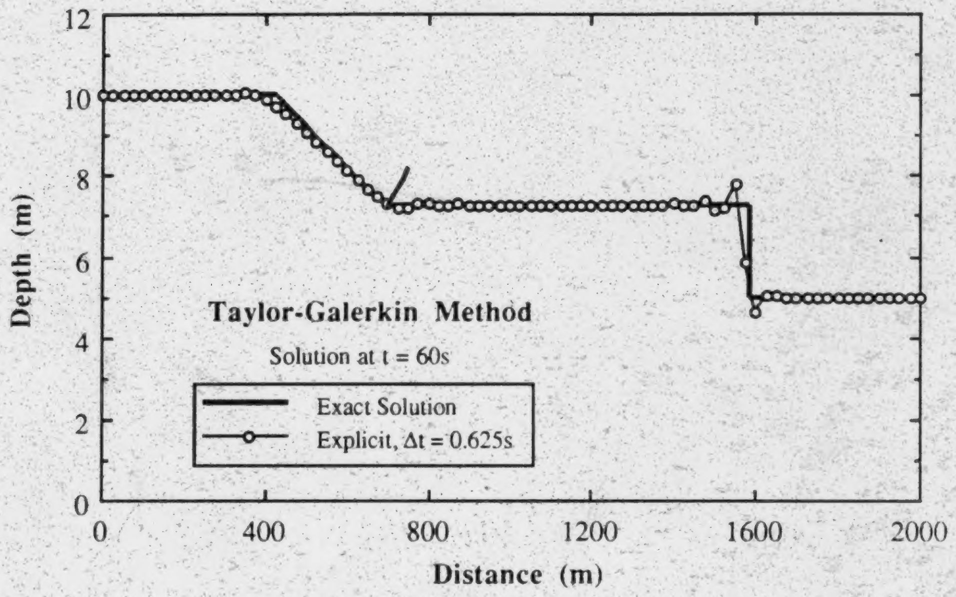
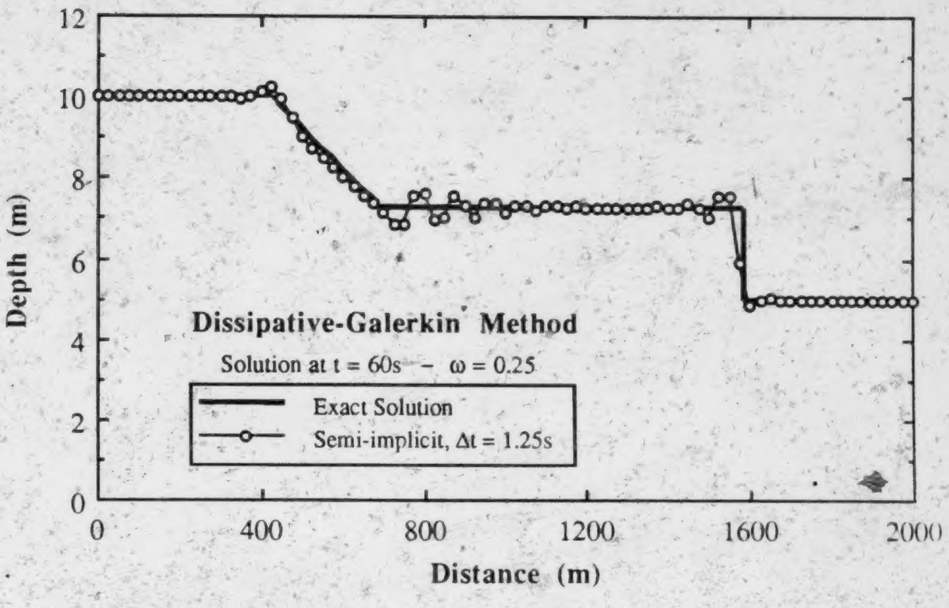
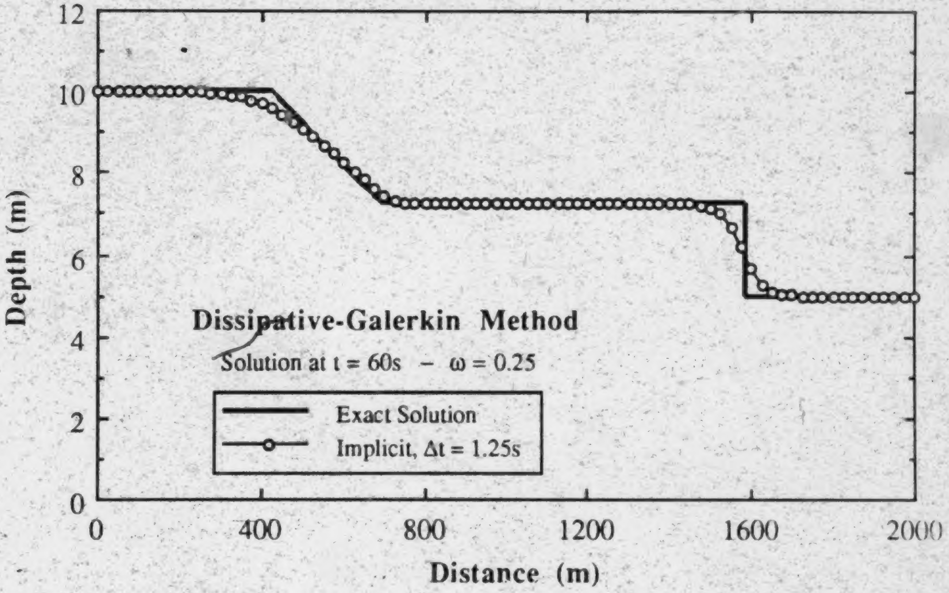


Figure 5.87 Dambreak, $H_1/H_o = 0.5$, (t = 60s), Taylor-Galerkin method, $\theta = 0.0$, $\Delta t = 0.625s$.

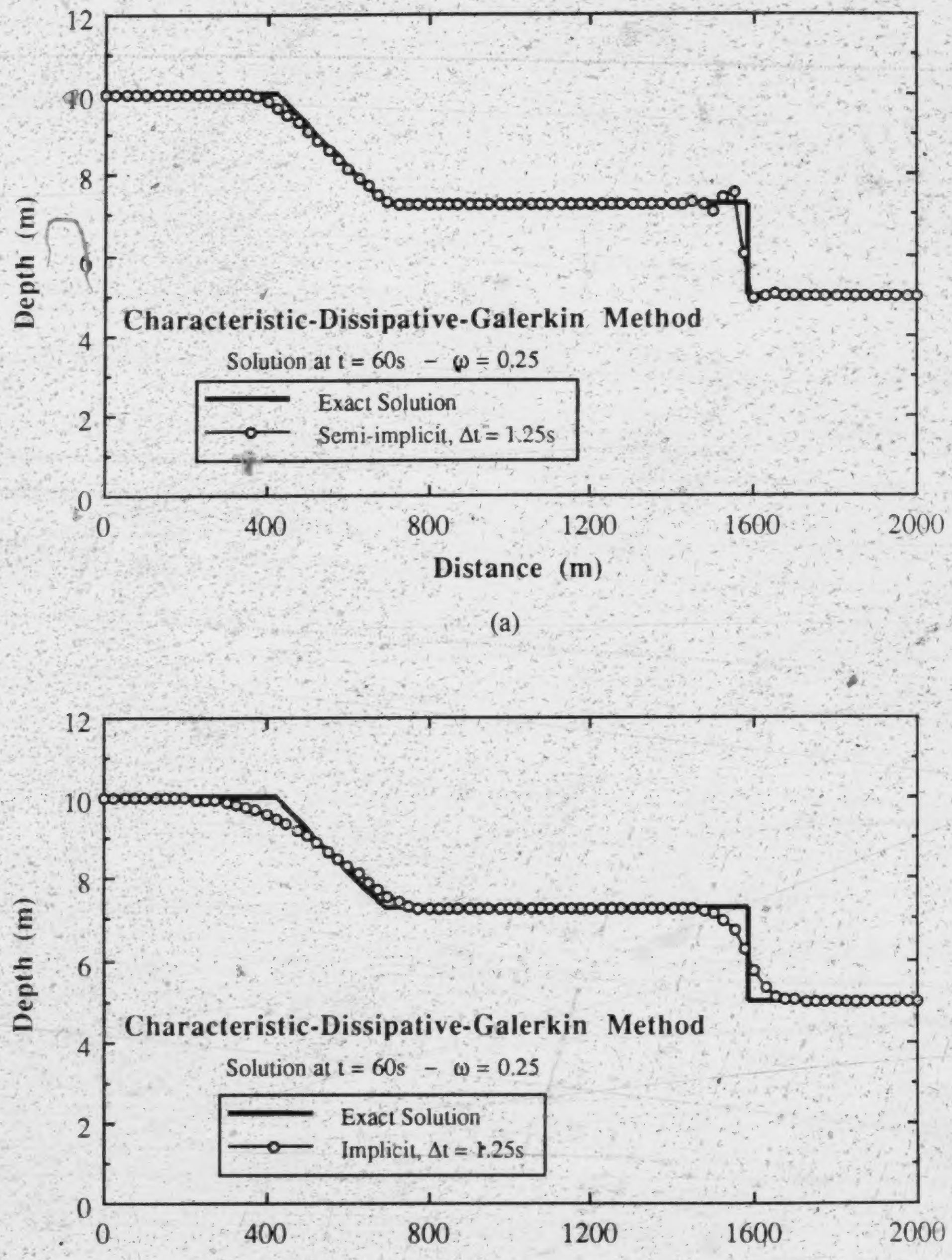


(a)



(b)

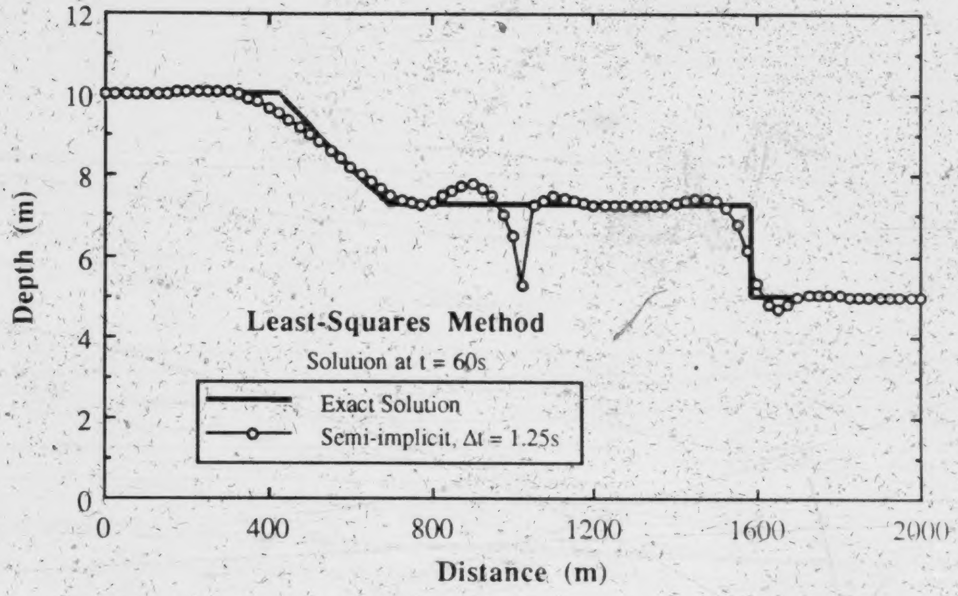
Figure 5.88 Dambreak, $H_1/H_o = 0.5$, ($t = 60$s), Dissipative-Galerkin method, $\omega = 0.25$, $\Delta t = 1.25$s: (a) $\theta = 0.5$; (b) $\theta = 1.0$.



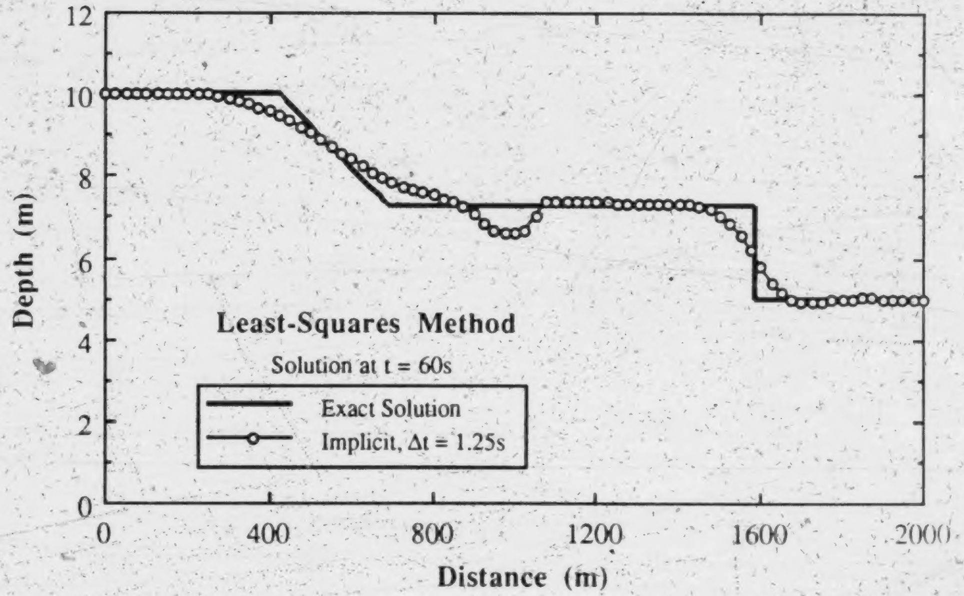
(a)

(b)

Figure 5.89 Dambreak, $H_1/H_o = 0.5$, (t = 0.60s), Characteristic-Dissipative-Galerkin method, $\omega = 0.25$, $\Delta t = 1.25s$: (a) $\theta = 0.5$; (b) $\theta = 1.0$.



(a)



(b)

Figure 5.90 Dambreak, $H_1/H_o = 0.5$, (t = 60s), Least-Squares method, $\Delta t = 1.25$s:
(a) $\theta = 0.5$; (b) $\theta = 1.0$.

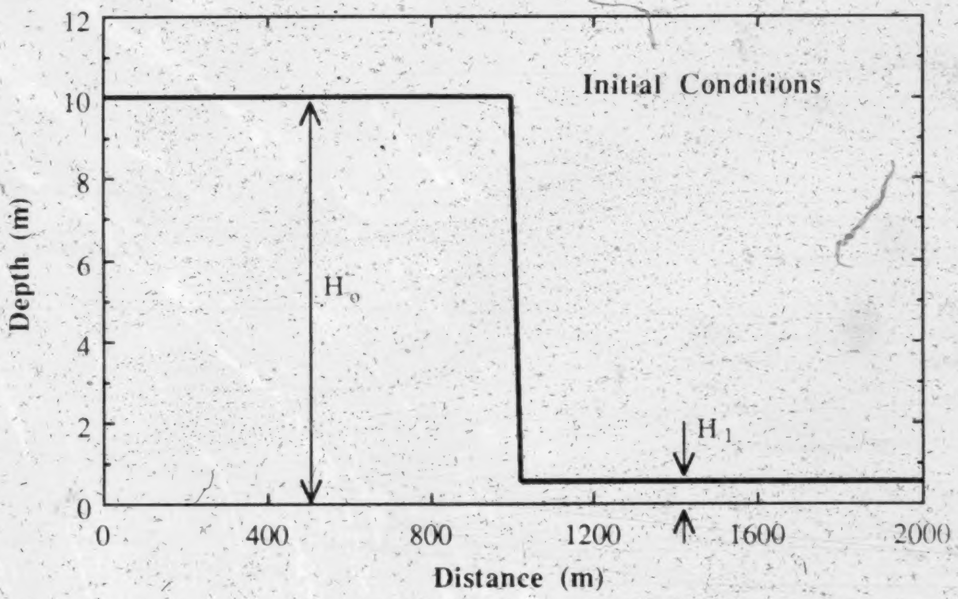
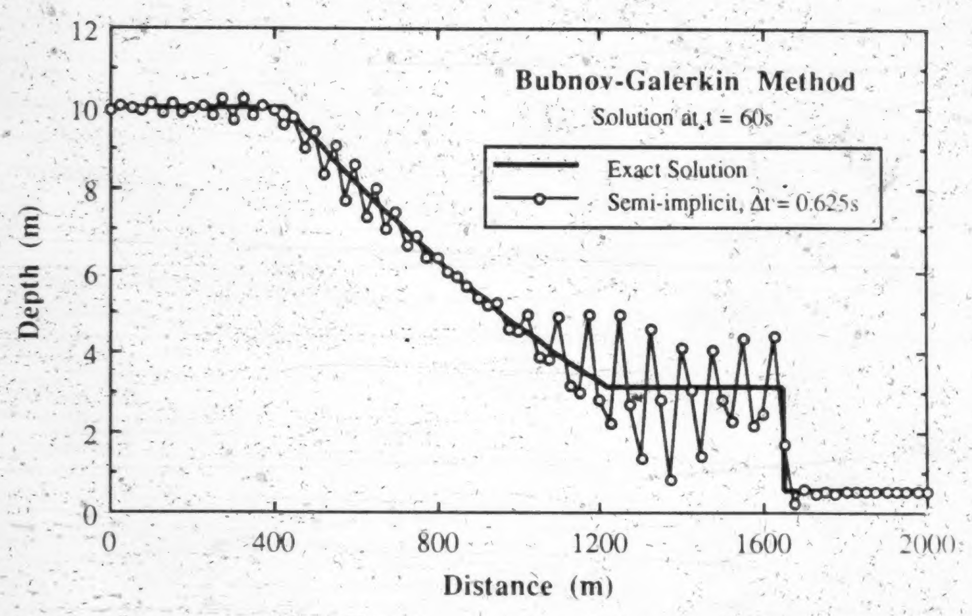
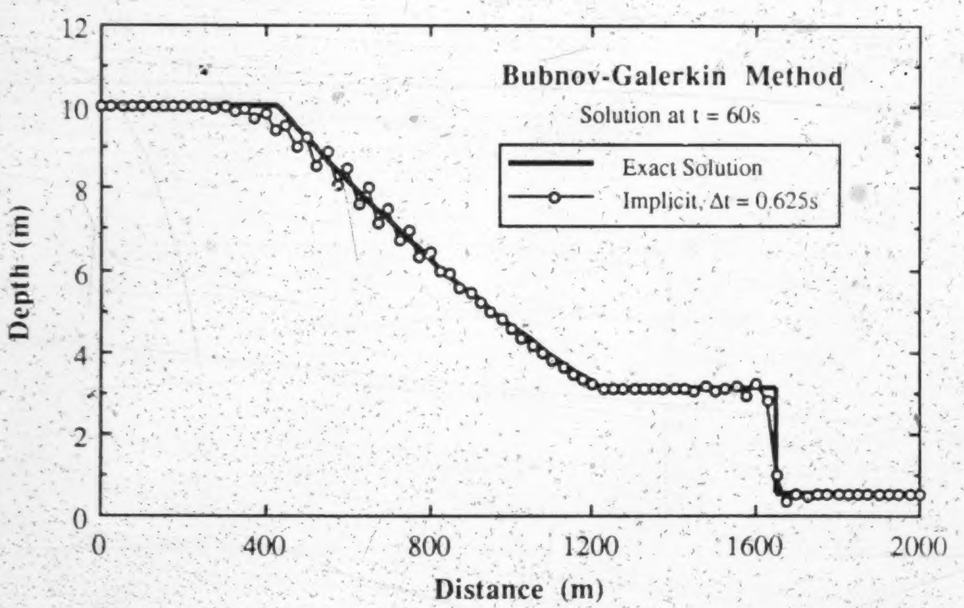


Figure 5.91 Dambreak, $H_1/H_0 = 0.05$, (t = 0s), initial conditions.

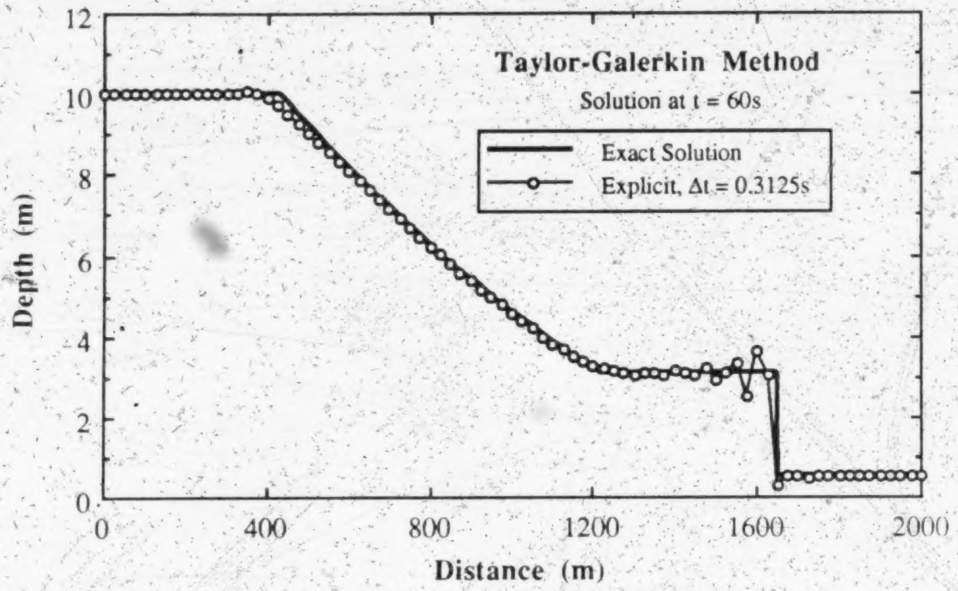


(a)

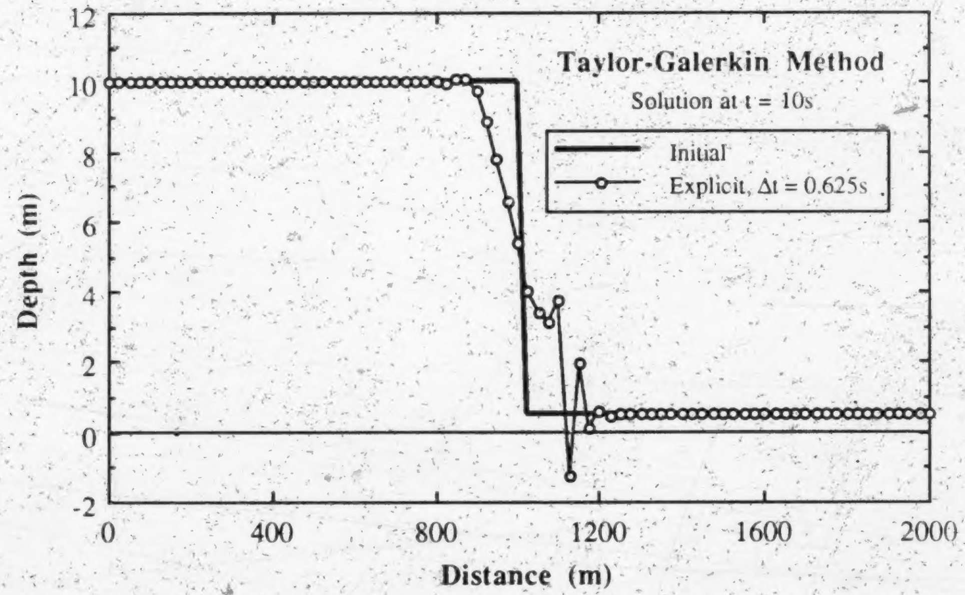


(b)

Figure 5.92 Dambreak, $H_1/H_0 = 0.05$, ($t = 60s$), Bubnov-Galerkin method, $\Delta t = 0.625s$:
(a) $\theta = 0.5$; (b) $\theta = 1.0$.

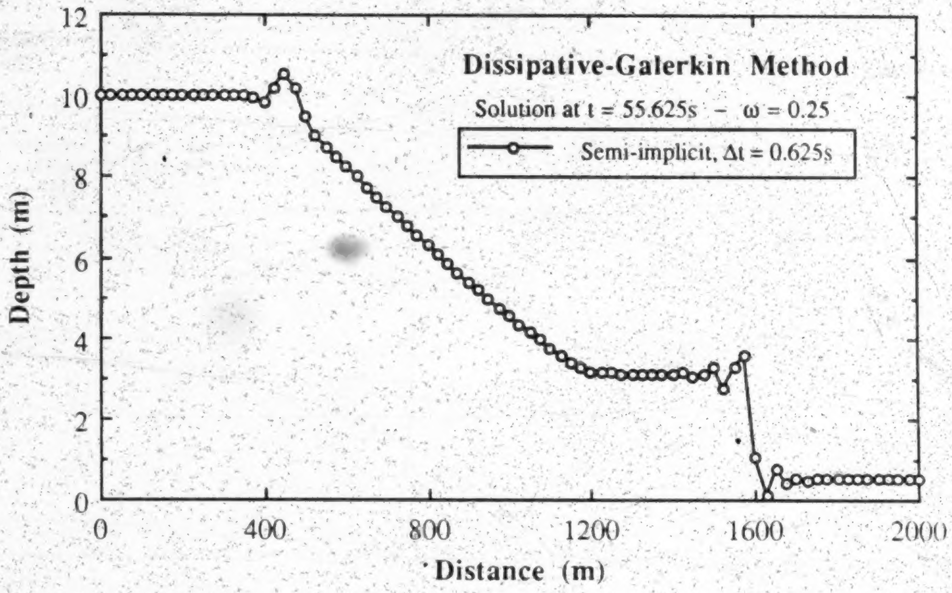


(a)

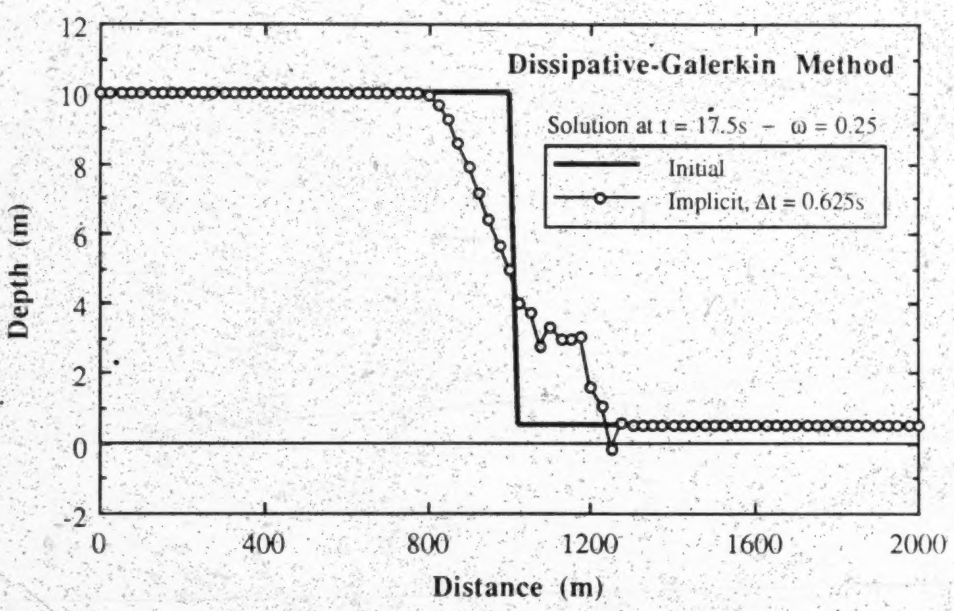


(b)

Figure 5.93 Dambreak, $H_1/H_0 = 0.05$, Taylor-Galerkin method, $\theta = 0.0$, (a) $\Delta t = 0.3125s$, $t = 60s$; (b) $\Delta t = 0.625s$, $t = 10s$.

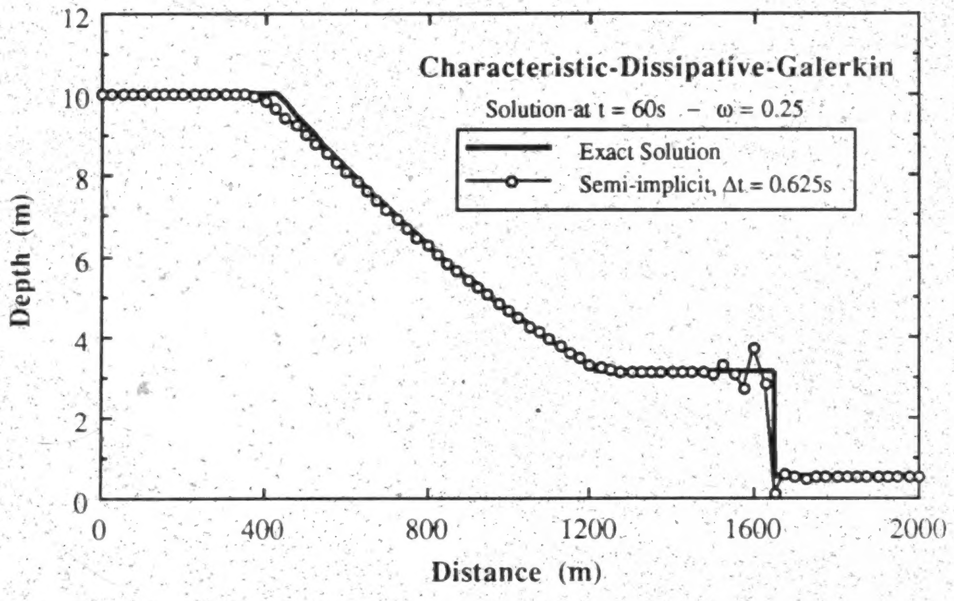


(a)

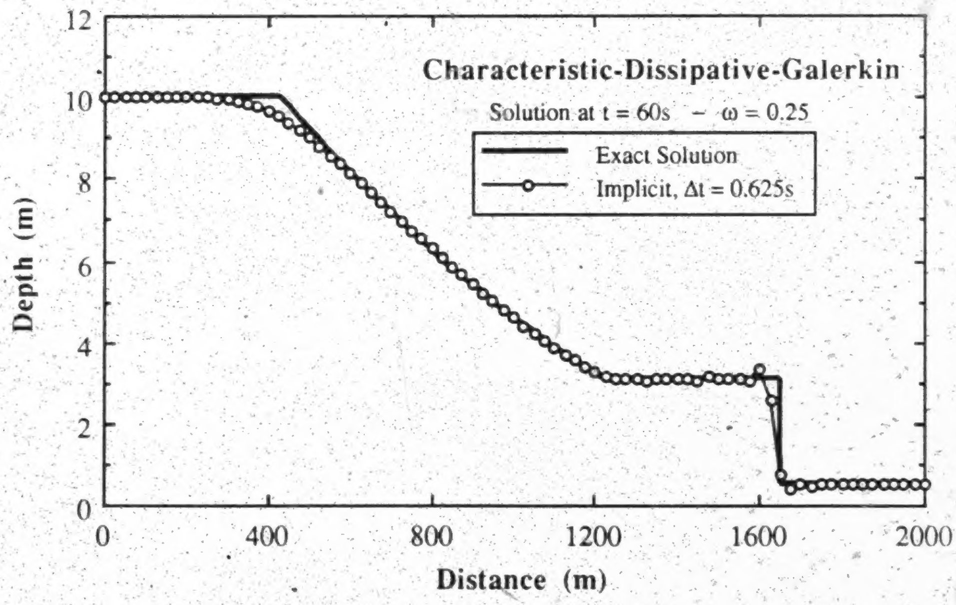


(b)

Figure 5.94 Dambreak, $H_1/H_0 = 0.05$, Dissipative-Galerkin method, $\omega = 0.25$, $\Delta t = 0.625$s: (a) $\theta = 0.5$, t =55.625s; (b) $\theta = 1.0$, t = 60s.

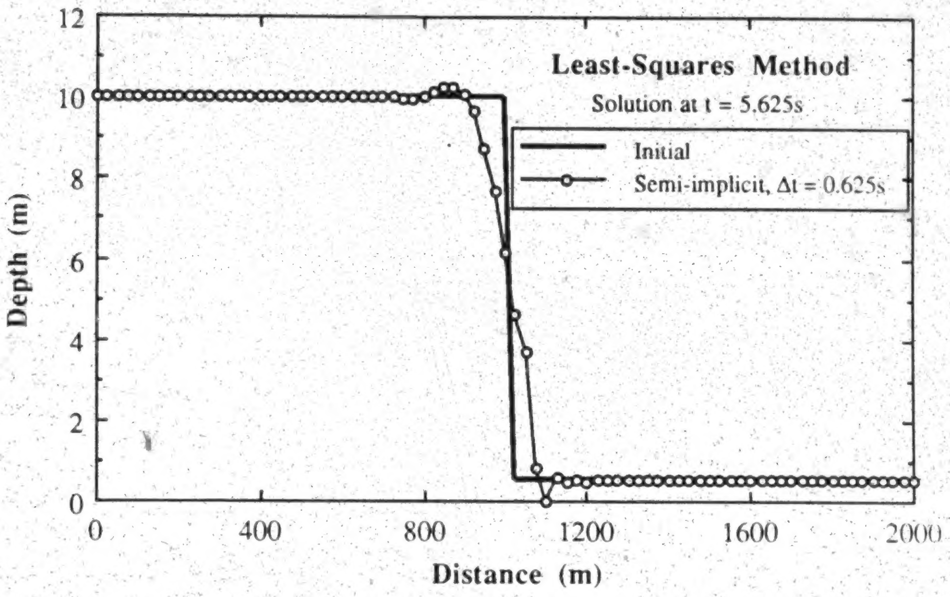


(a)

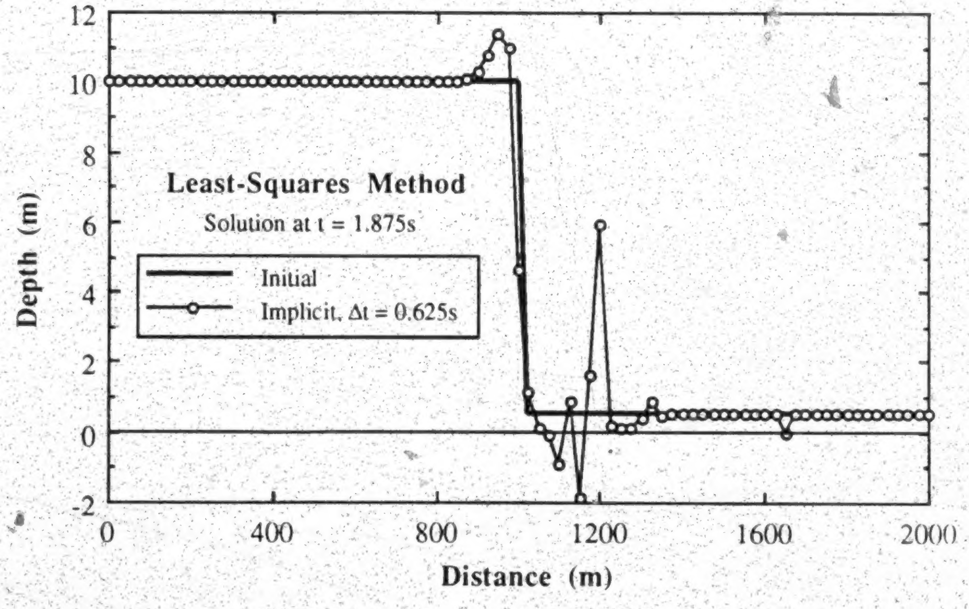


(b)

Figure 5.95 Dambreak, $H_1/H_0 = 0.05$, ($t = 0.60$s), Characteristic-Dissipative-Galerkin method, $\omega = 0.25$; $\Delta t = 0.625$s: (a) $\theta = 0.5$; (b) $\theta = 1.0$.

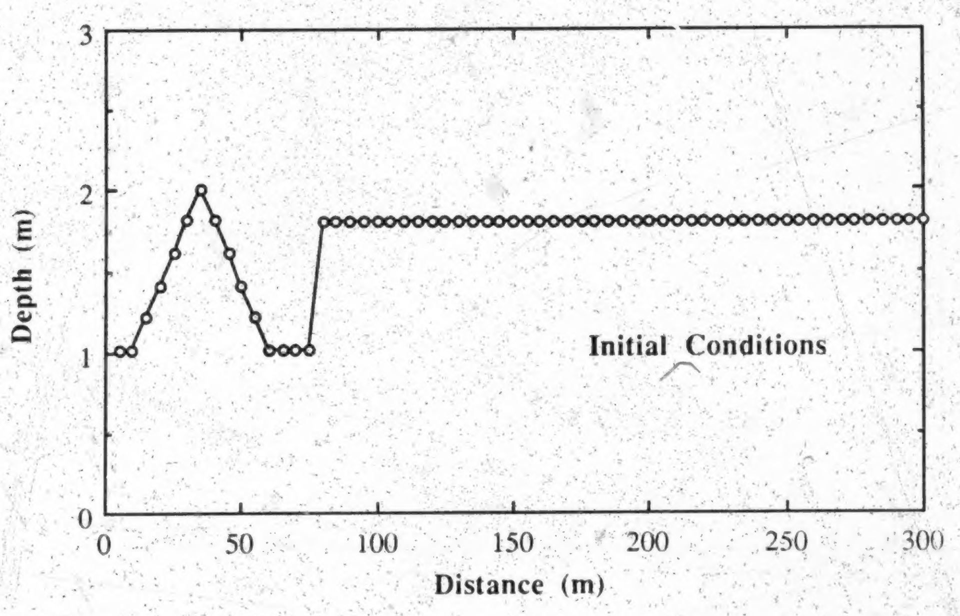


(a)



(a)

Figure 5.96 Dambreak, $H_1/H_0 = 0.05$, Least-Squares method, $\Delta t = 0.625$s: (a) $\theta = 0.5$, $t = 60$s; (b) $\theta = 1.0$, $t = 60$s.



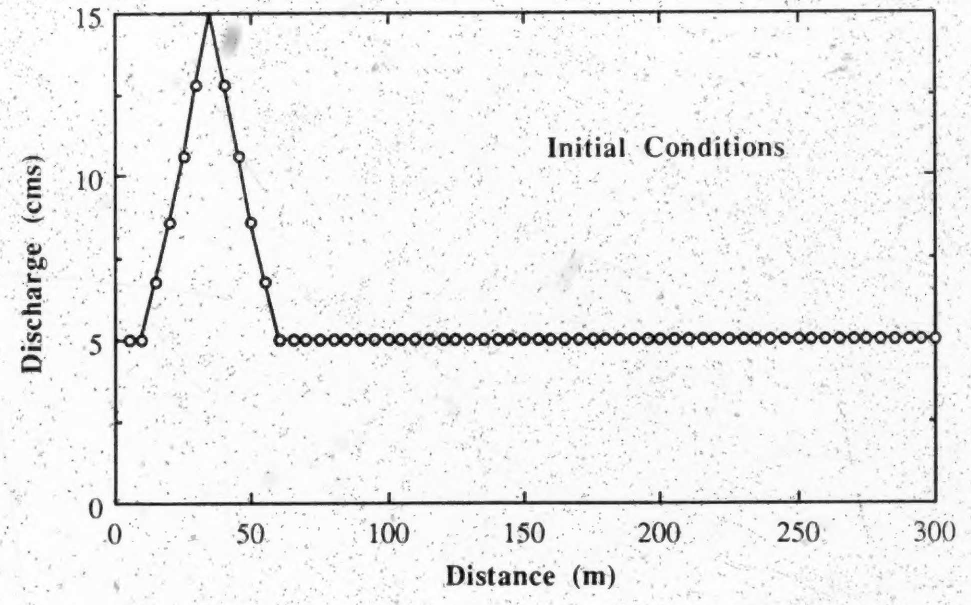
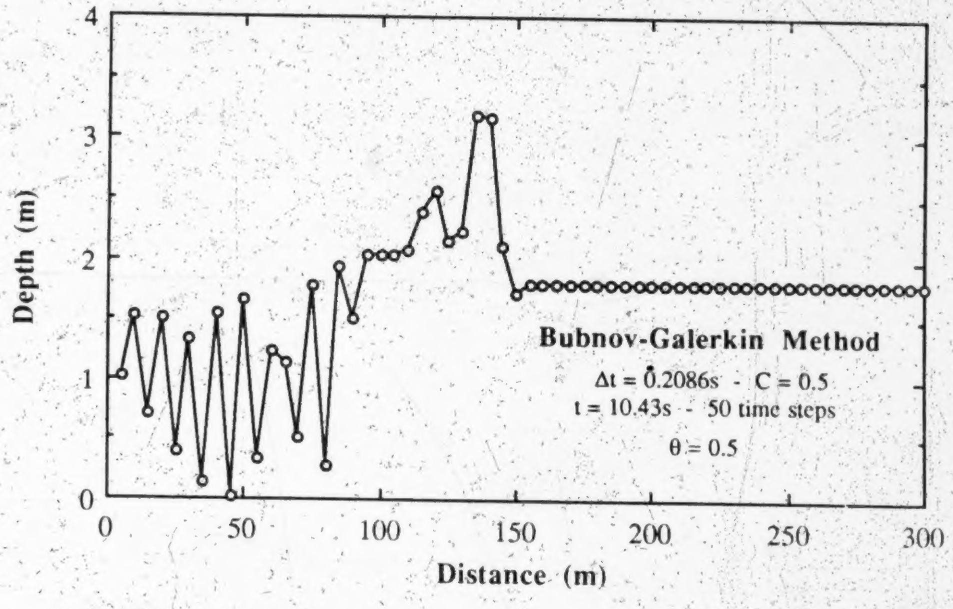


Figure 5.97 Propagation of a wave through an hydraulic jump (t = 0s), initial conditions.



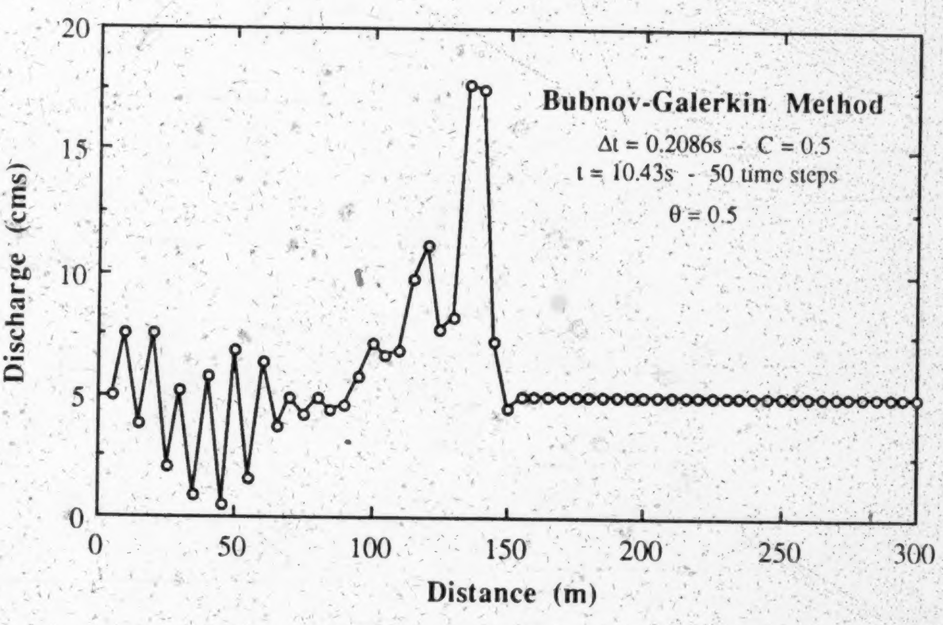
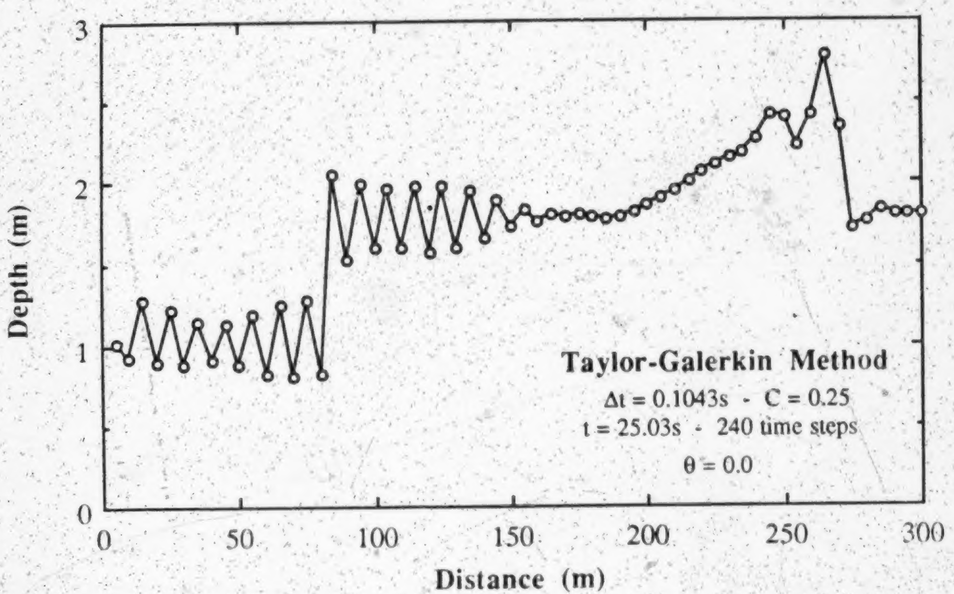


Figure 5.98 Propagation of a wave through an hydraulic jump (t = 10.43s), Bubnov-Galerkin method, θ = 0.5, C = 0.5, Δt = 0.2086s.



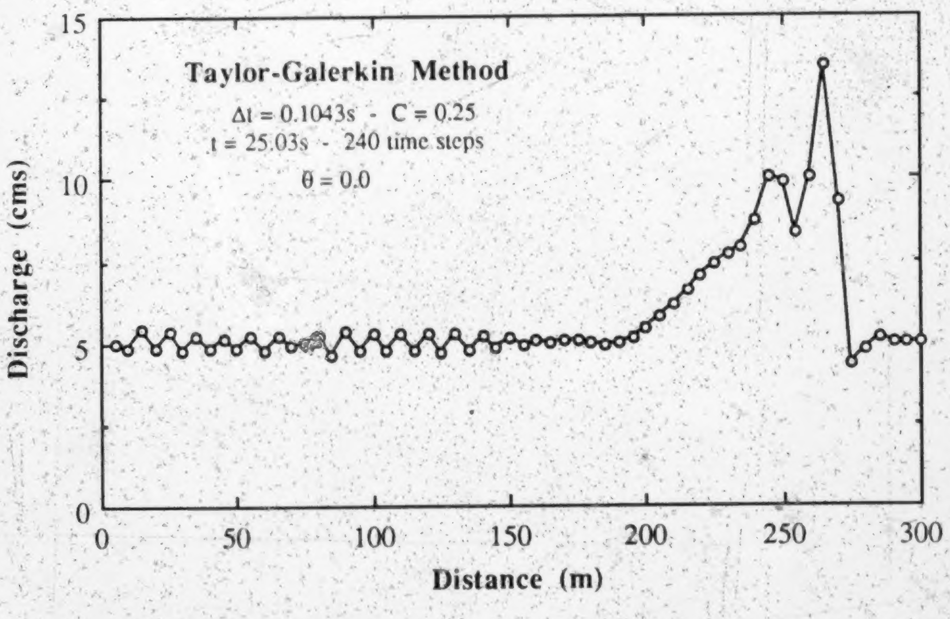


Figure 5.99 Propagation of a wave through an hydraulic jump (t = 25.03s), Taylor-Galerkin method, θ = 0.0, C = 0.25, Δt = 0.1043s.

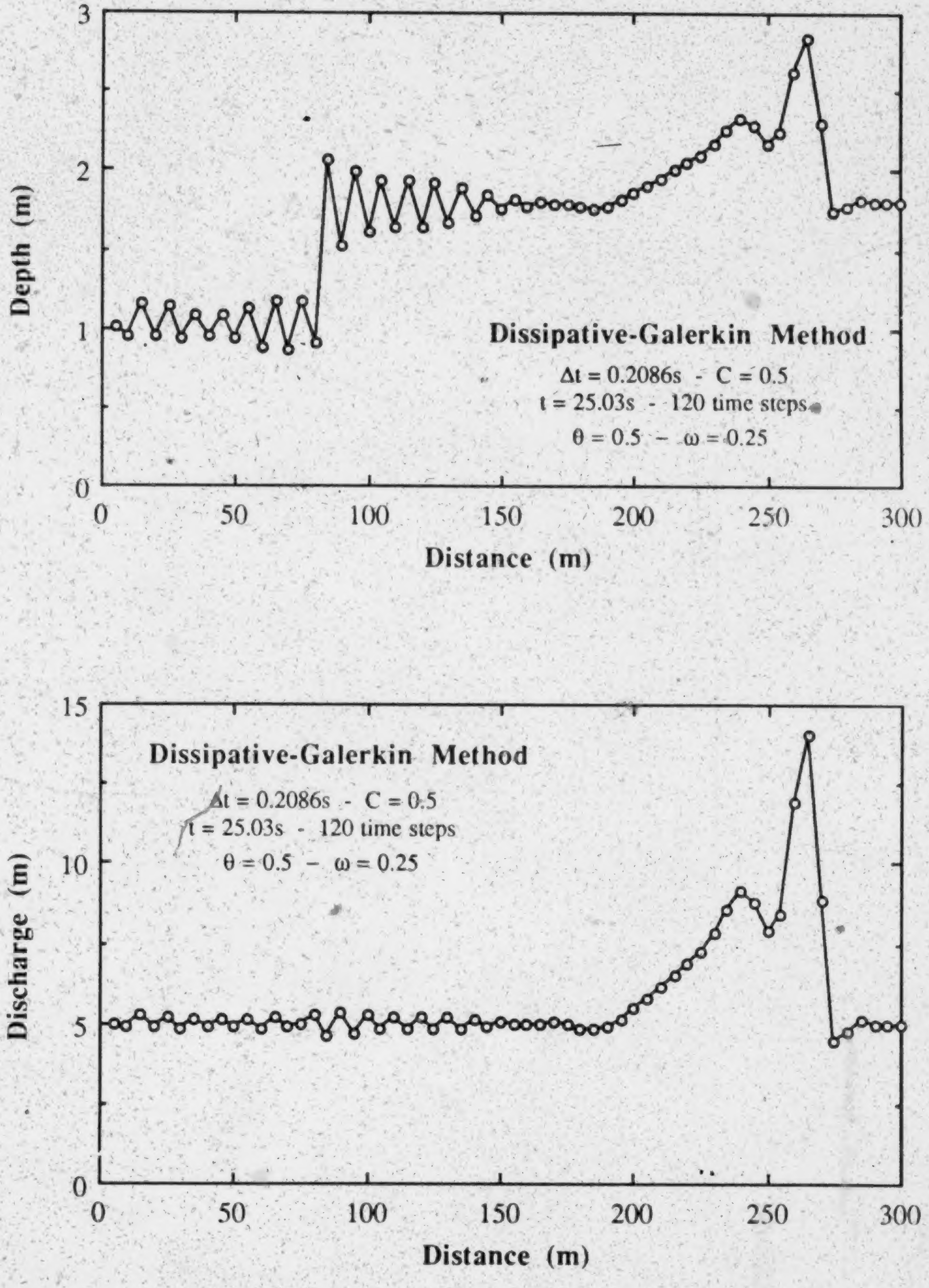
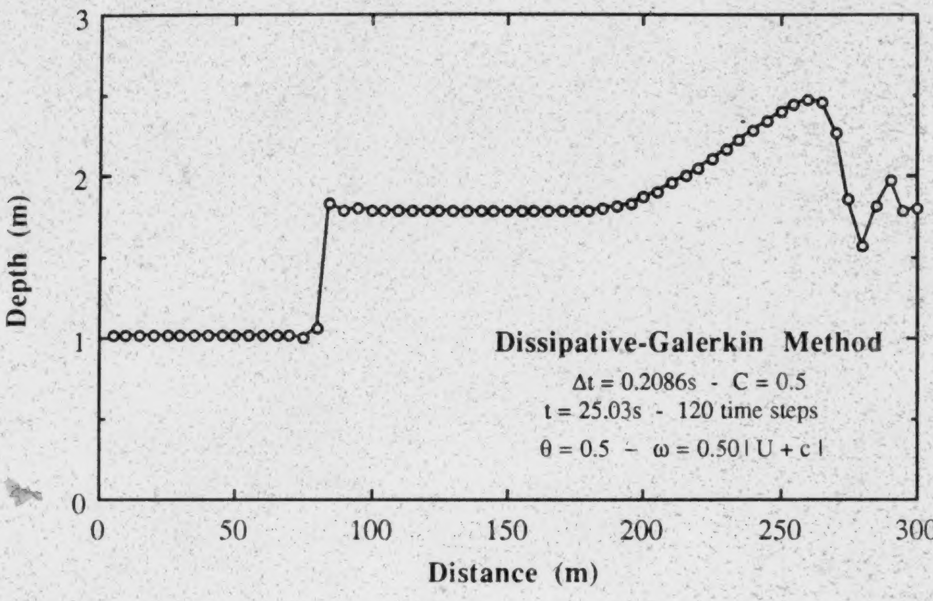


Figure 5.100 Propagation of a wave through an hydraulic jump (t = 25.03s), Dissipative-Galerkin method, θ = 0.5, C = 0.5, Δt = 0.2086s, ω = 0.25.



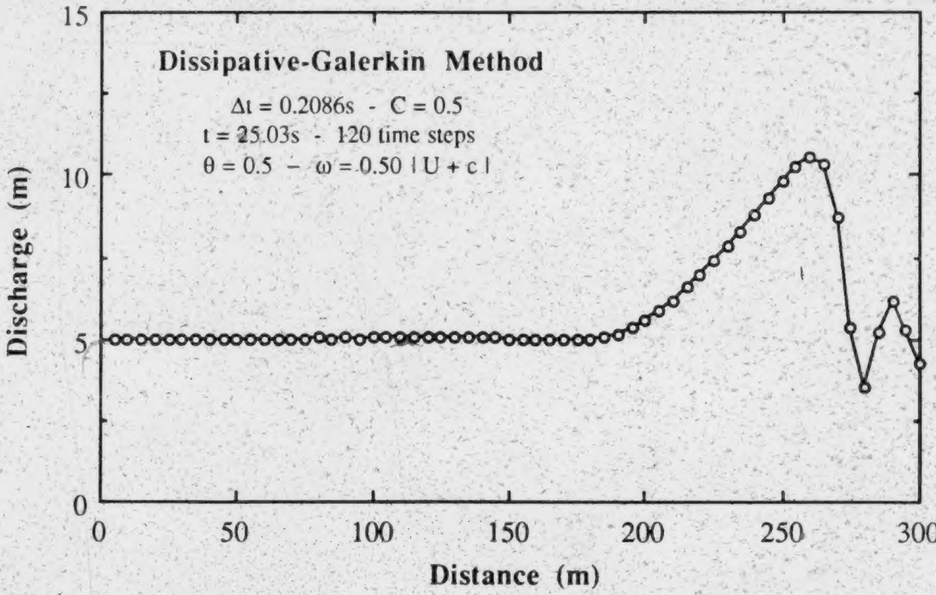
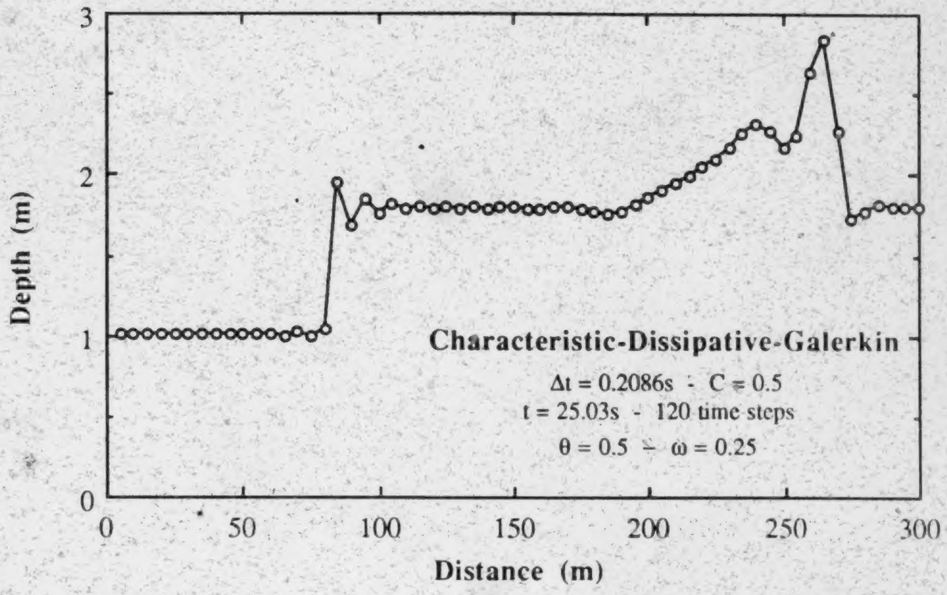


Figure 5.101 Propagation of a wave through an hydraulic jump (t = 25.03s), Dissipative-Galerkin method, θ = 0.5, C = 0.5, Δt = 2086s, ω = 0.50 | U+c |.



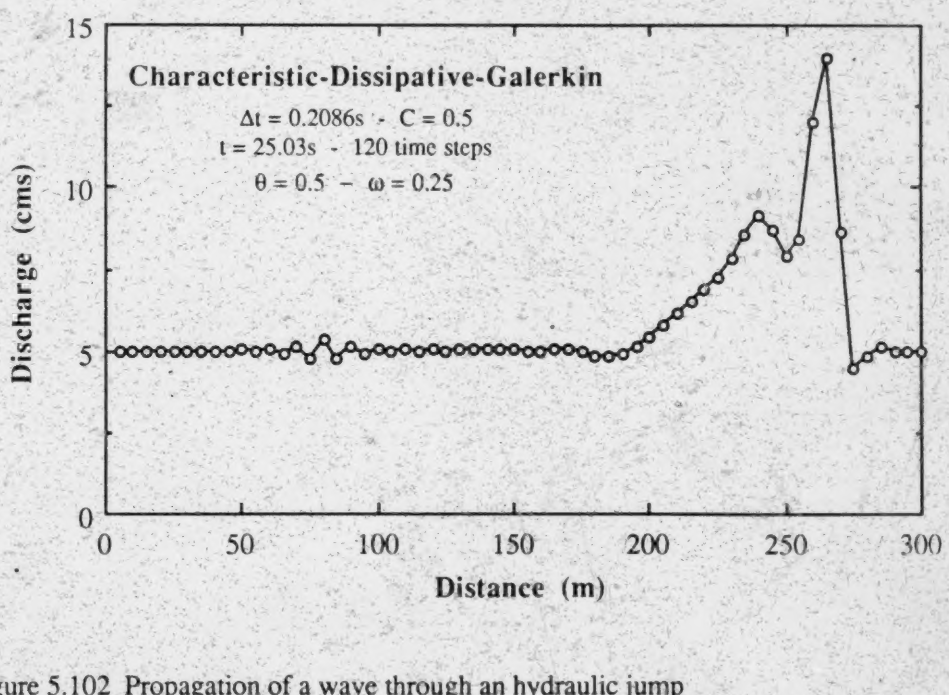
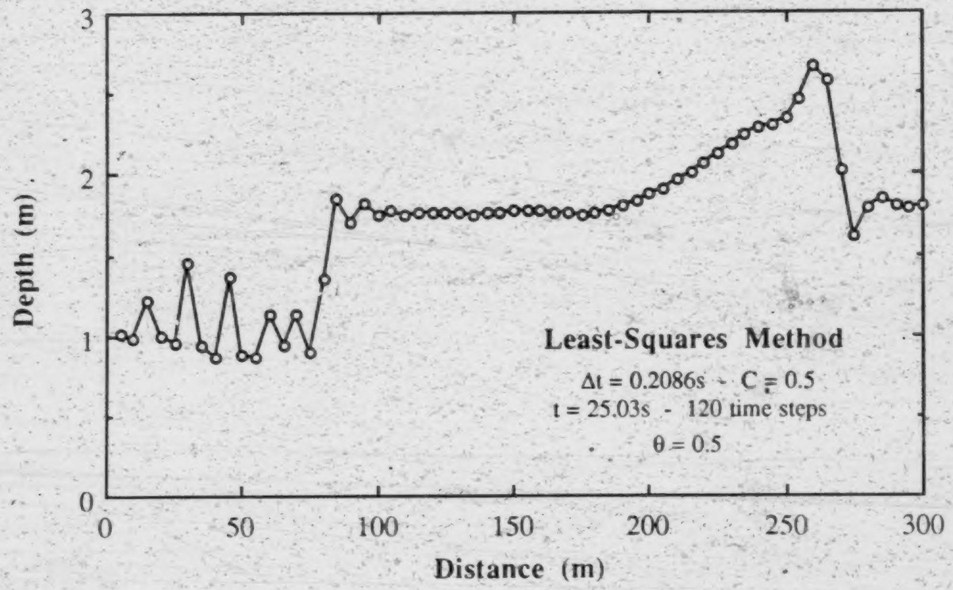


Figure 5.102 Propagation of a wave through an hydraulic jump (t = 25.03s), Characteristic-Dissipative-Galerkin method, θ = 0.5, C = 0.5, Δt = 0.2086s, ω = 0.25.



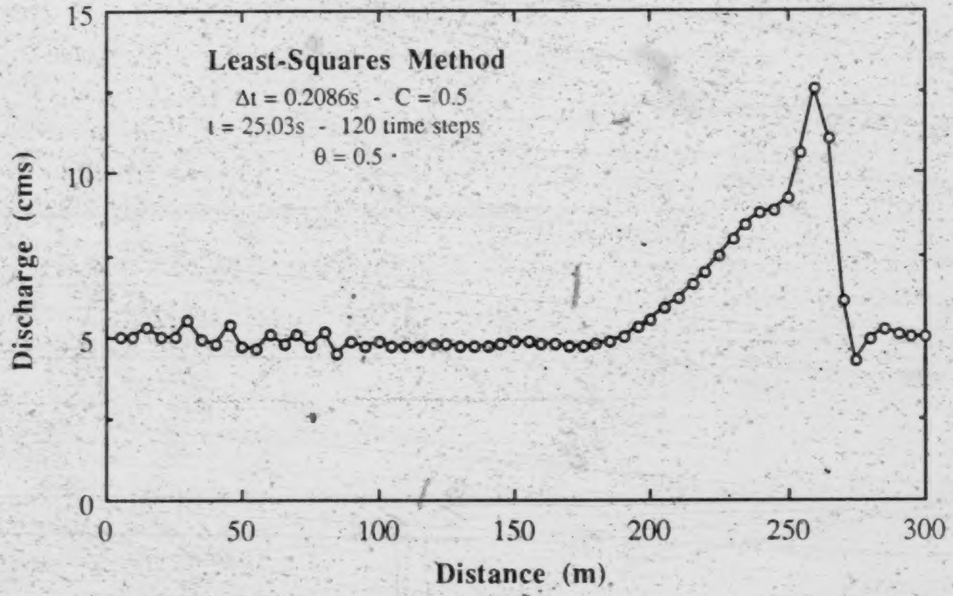
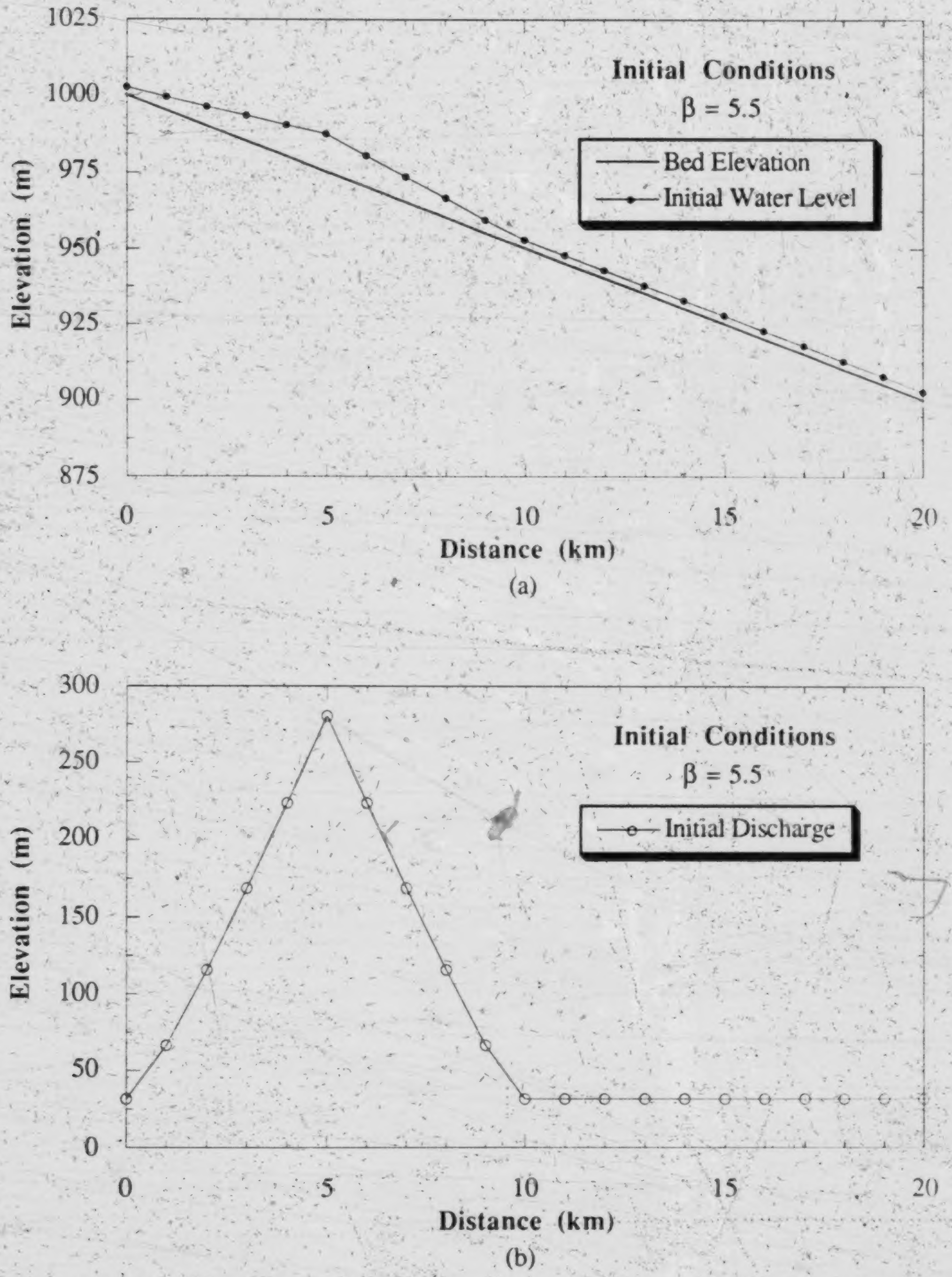


Figure 5.103 Propagation of a wave through an hydraulic jump (t = 25.03s), Least-Squares method, θ = 0.5, C = 0.5, Δt = 0.2086s.



(a)

(b)

Figure 5.104 Kinematic wave test, $\beta = 5.5$, initial conditions.

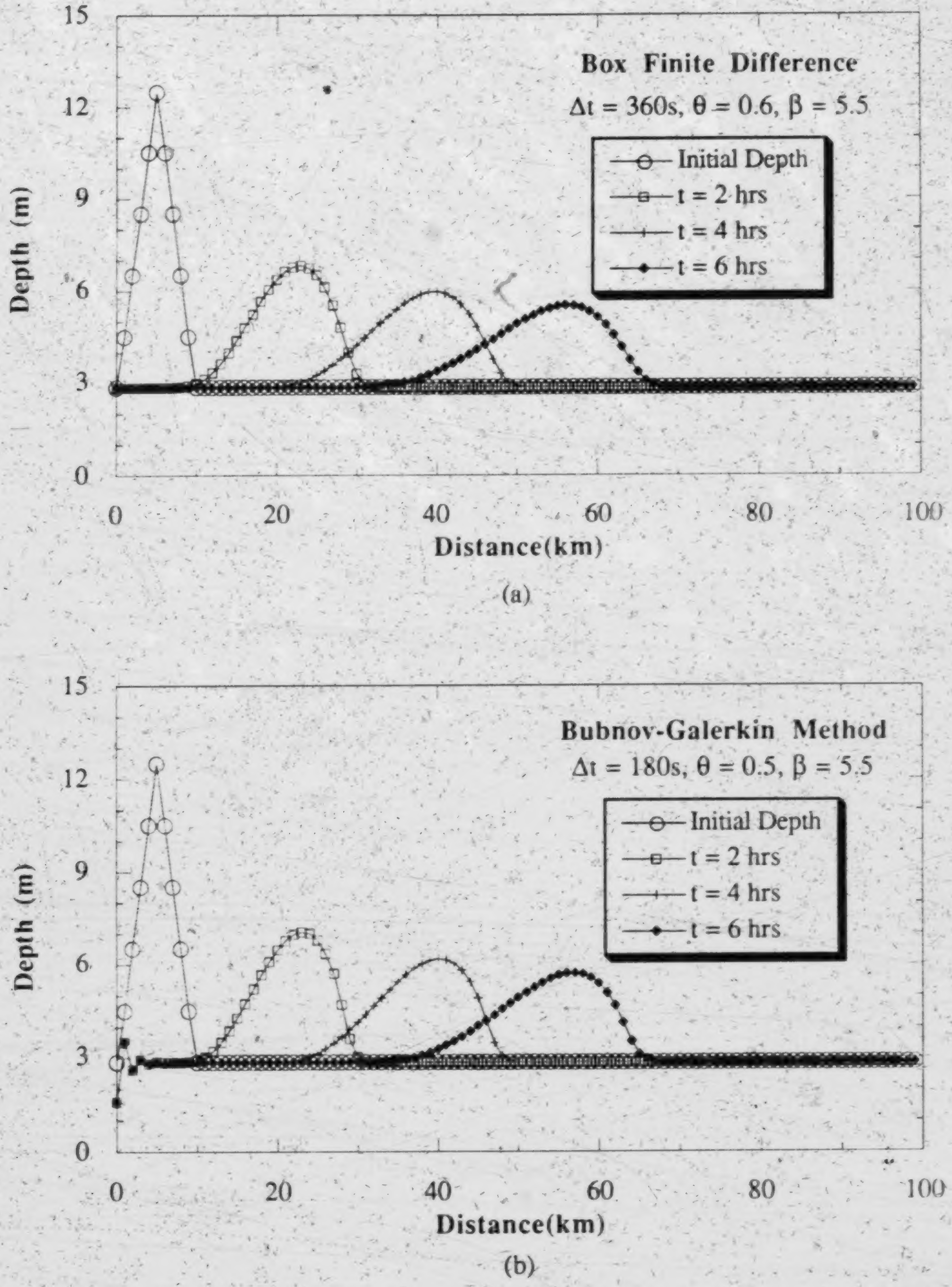
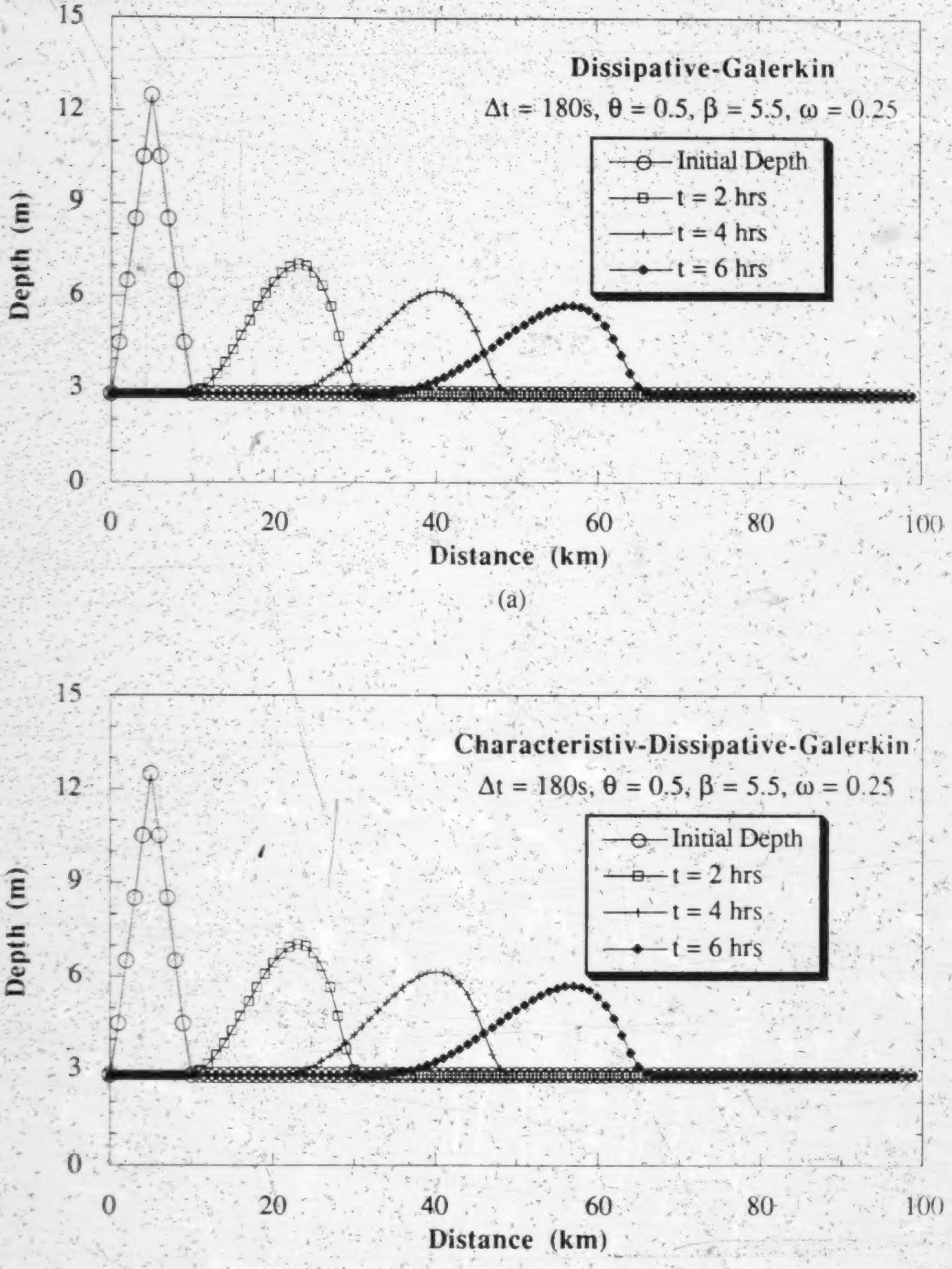


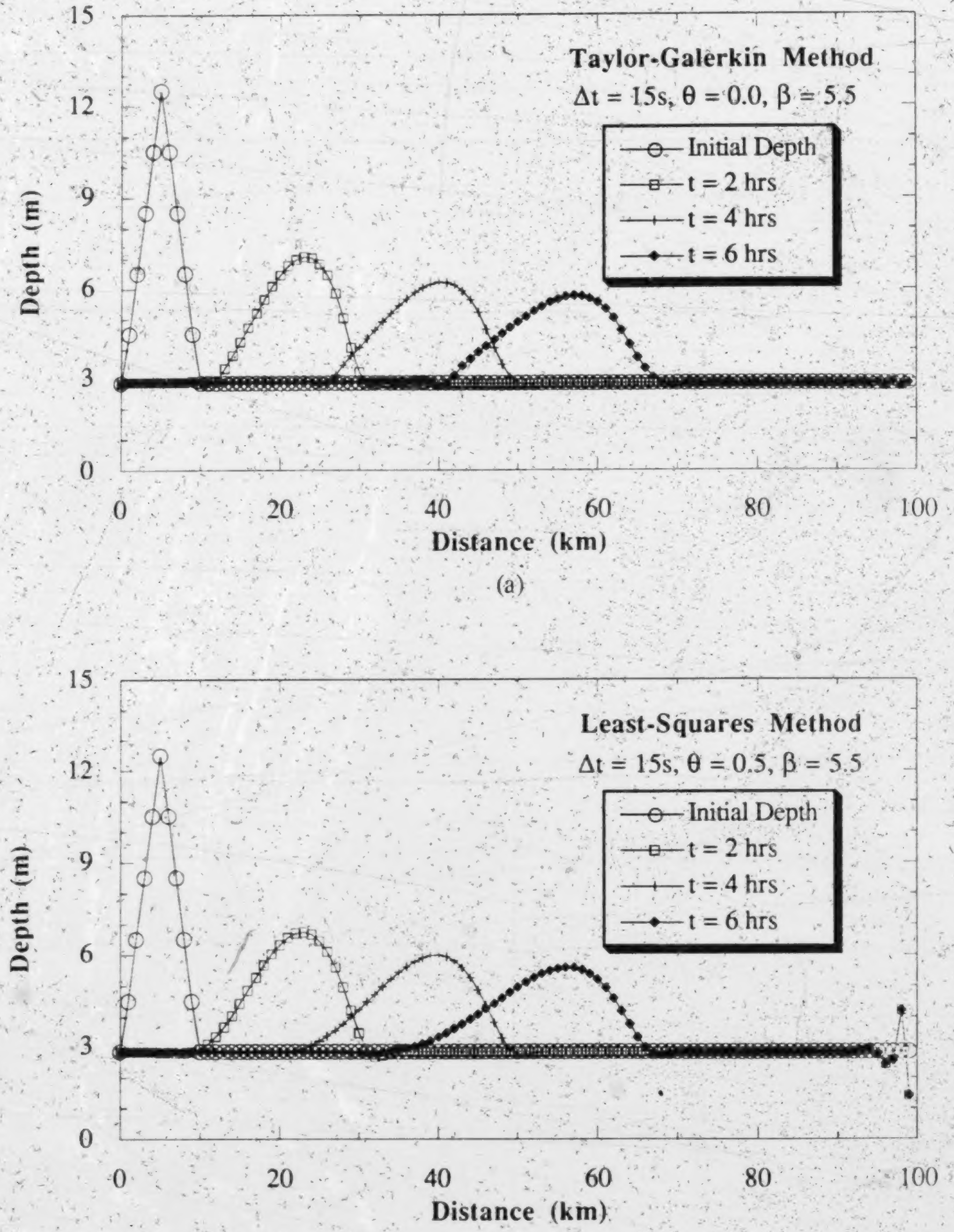
Figure 5.105 Kinematic wave test, $\beta = 5.5$, (a) Box Finite Difference Method, $\theta = 0.6$, $\Delta t = 360s$; (b) Bubnov-Galerkin method, $\theta = 0.5$, $\Delta t = 180s$.



(a)

(b)

Figure 5.106 Kinematic wave test, β = 5.5, Δt = 180s, θ = 0.5, ω = 0.25, (a) Dissipative-Galerkin method; (b) Characteristic-Dissipative-Galerkin method.



(a)

(b)

Figure 5.107 Kinematic wave test, $\beta = 5.5$, $\Delta t = 15$s, (a) Taylor-Galerkin method, $\theta = 0.0$; (b) Least-Squares method, $\theta = 0.5$.

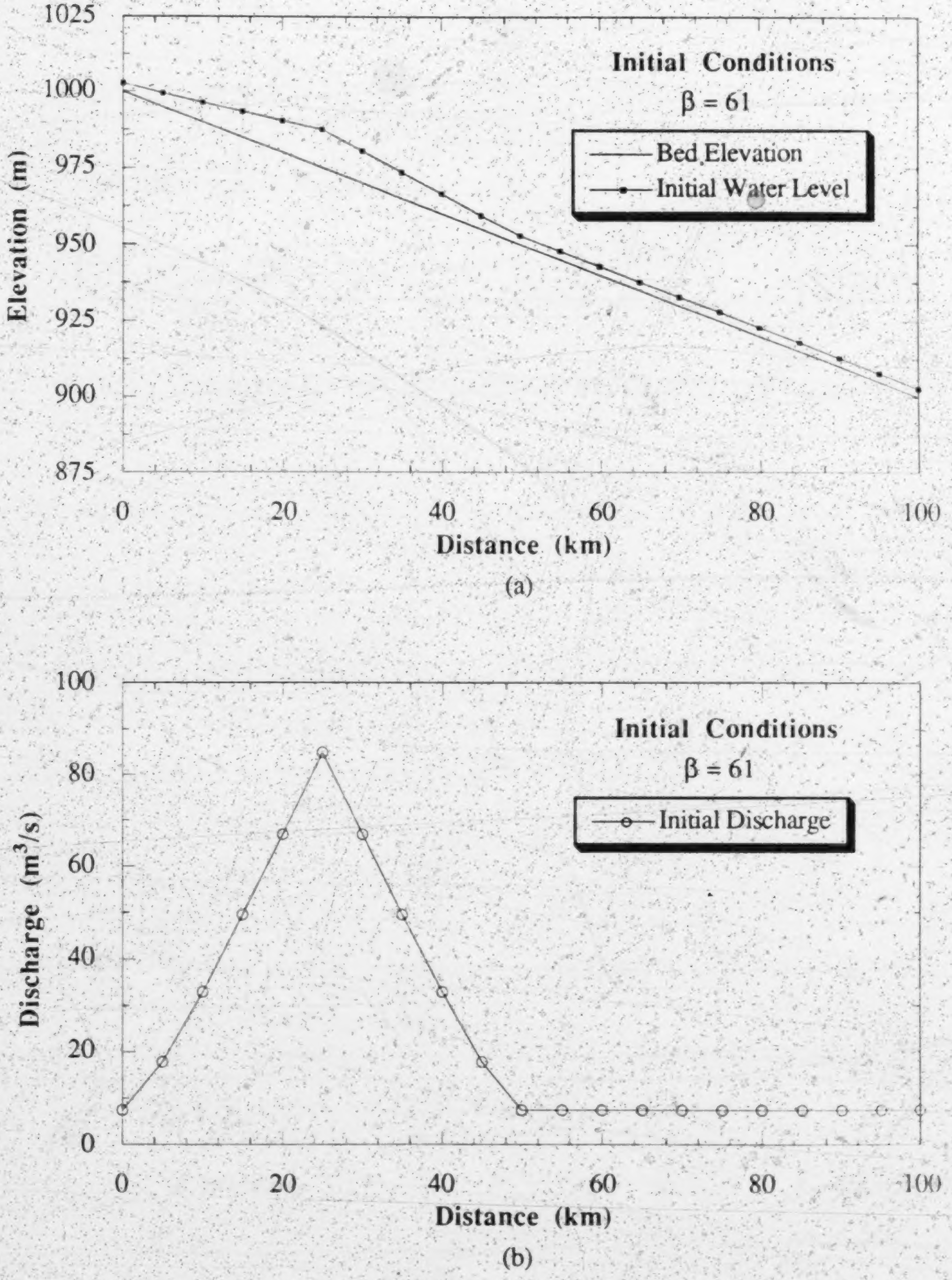
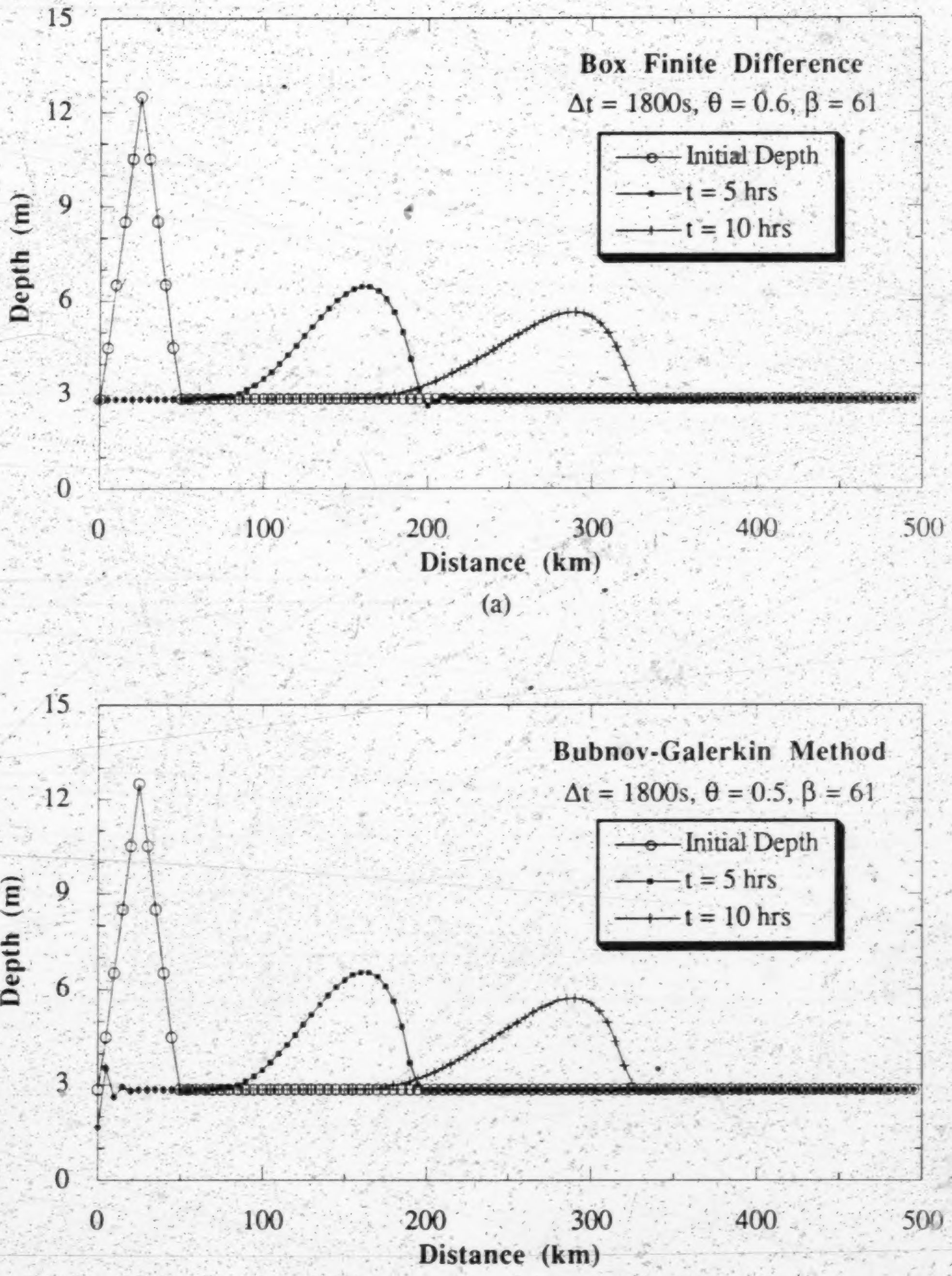


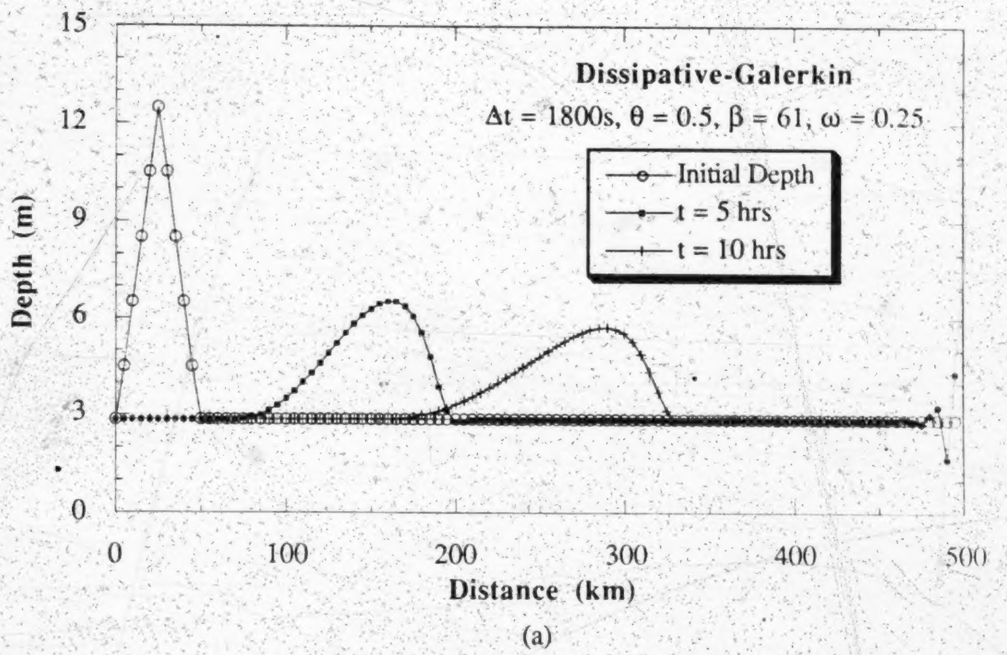
Figure 5.108 Kinematic wave test, $\beta = 61$, initial conditions.



(a)

(b)

Figure 5.109 Kinematic wave test, β = 61, Δt = 1800s, (a) Box Finite Difference Method, θ = 0.6; (b) Bubnov-Galerkin method, θ = 0.5.



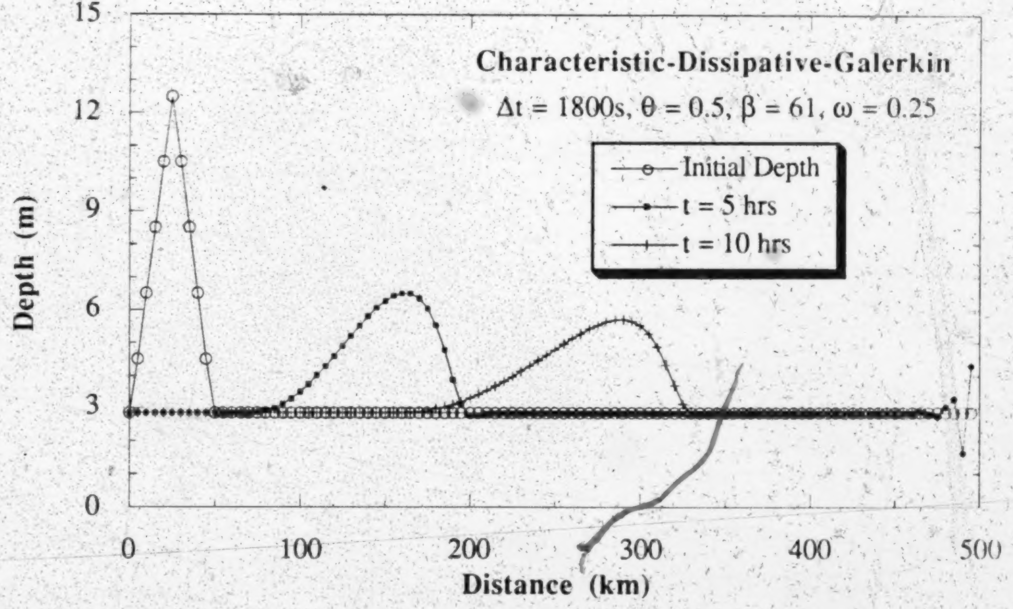
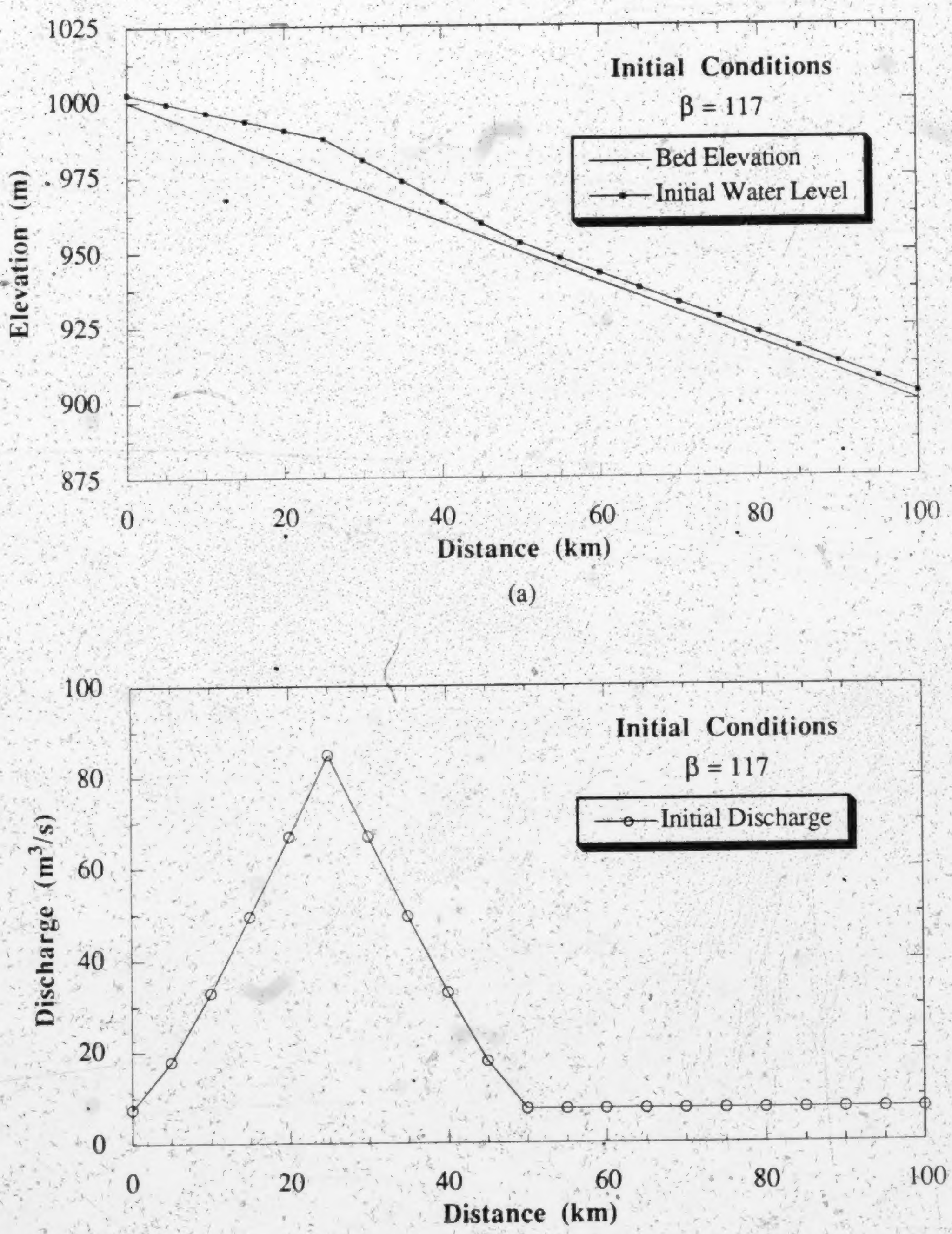


Figure 5.110 Kinematic wave test, β = 61, Δt = 1800s, θ = 0.5, ω = 0.25, (a) Dissipative-Galerkin method; (b) Characteristic-Dissipative-Galerkin method.



(a)

(b)

Figure 5.111 Kinematic wave test, $\beta = 117$, initial conditions.

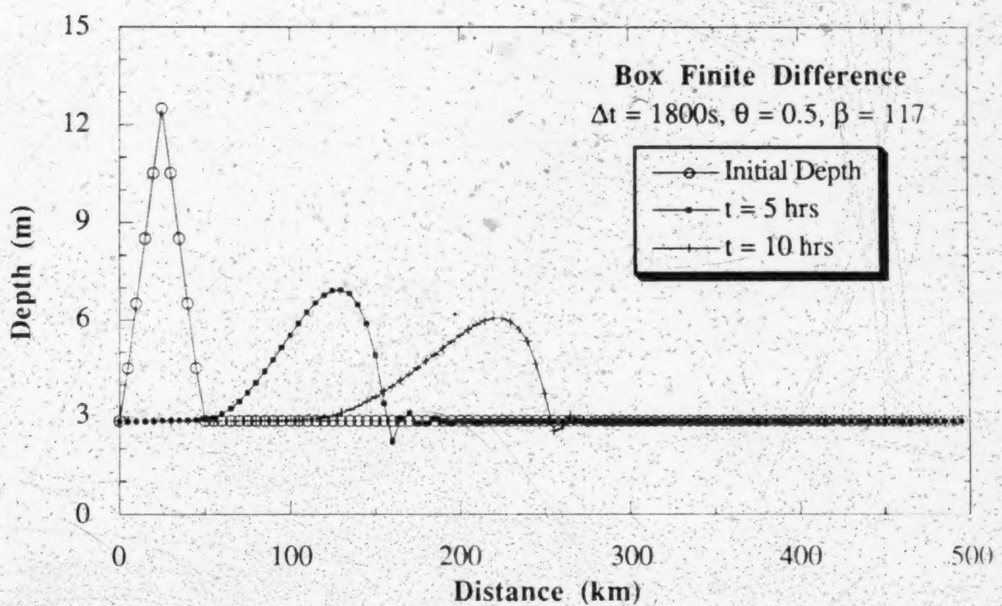
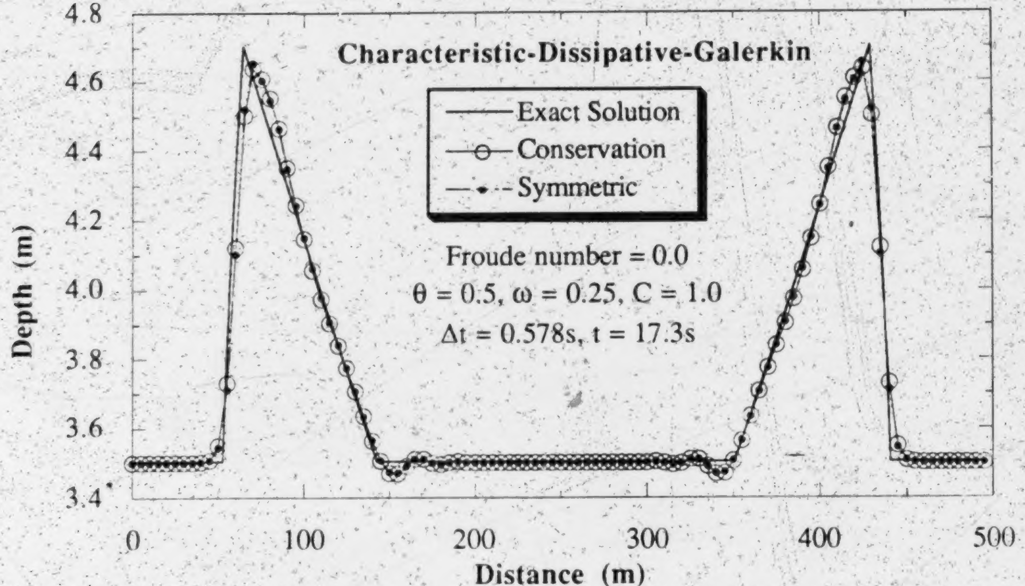
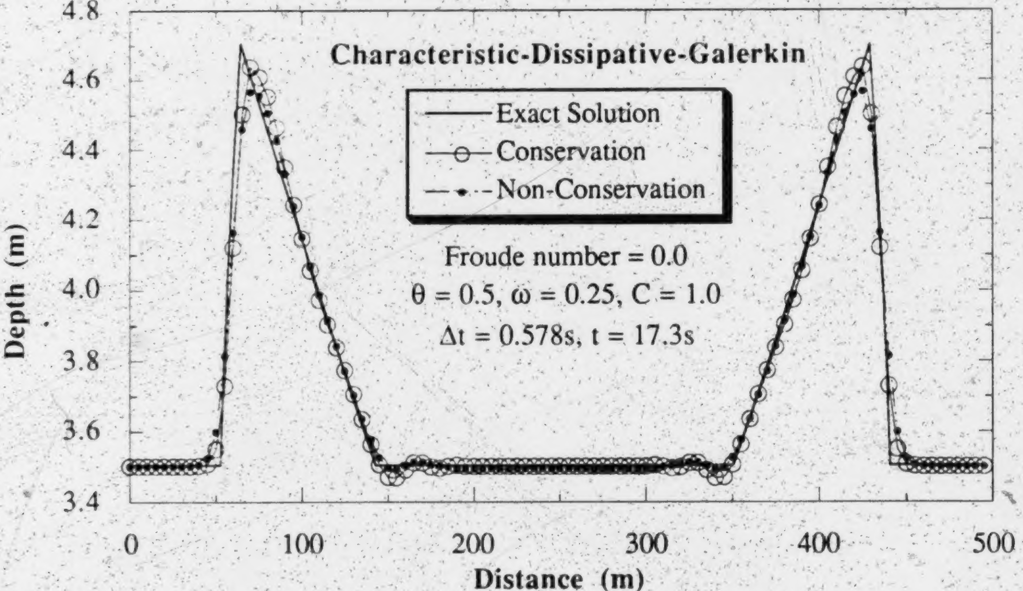


Figure 5.112 Kinematic wave test, $\beta = 117$, $\Delta t = 1800$s, $\theta = 0.6$, Box-Finite-Difference method.



(a)



(b)

Figure 5.113 Non-linear perturbation (t = 17.3s), Characteristic-Dissipative-Galerkin method, F = 0.0, θ = 0.5, ω = 0.25, C = 1.0, Δt = 0.578s (a) conservation form of the equations vs. symmetric form; (b) conservation form of the equations vs. non-conservation form.

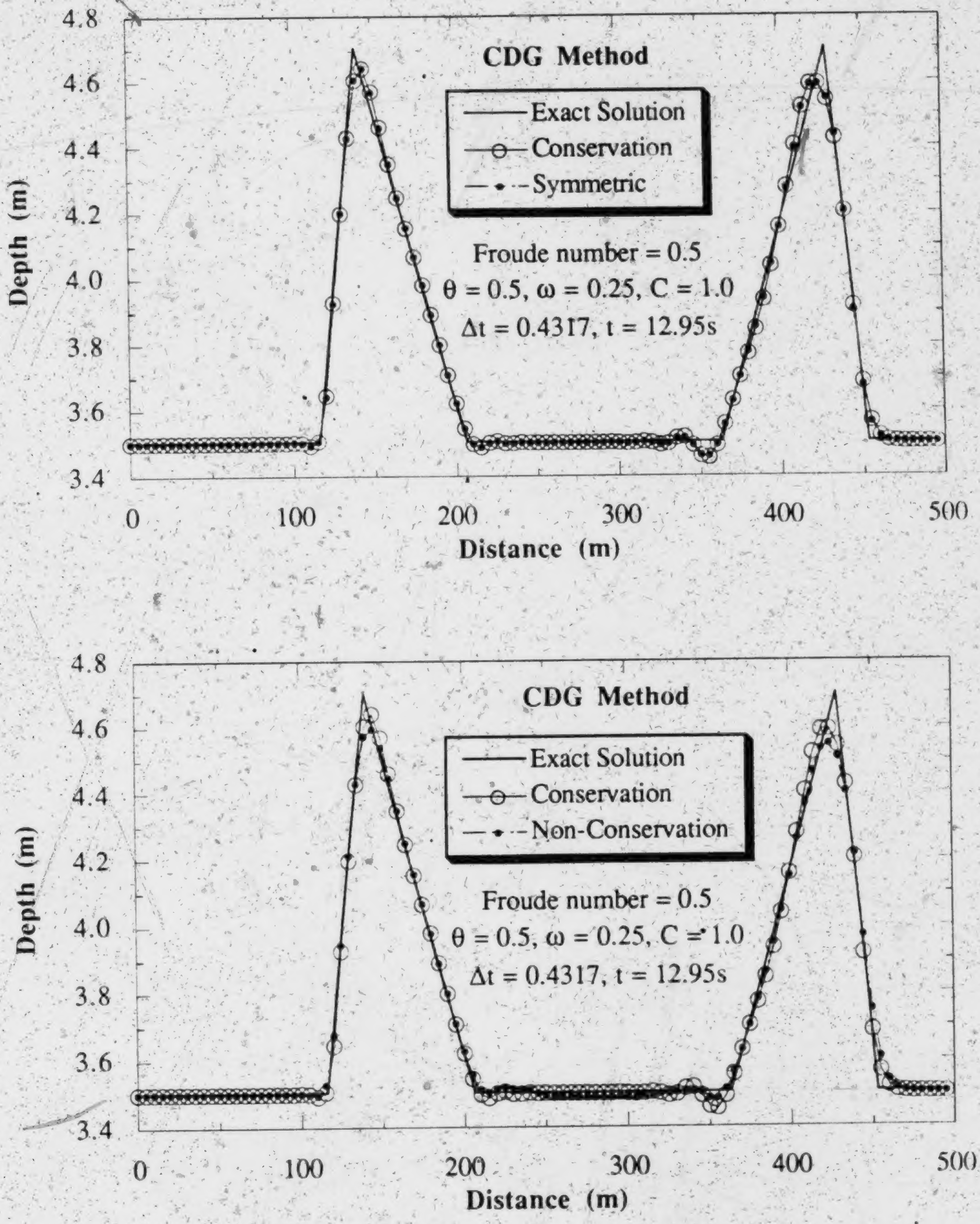
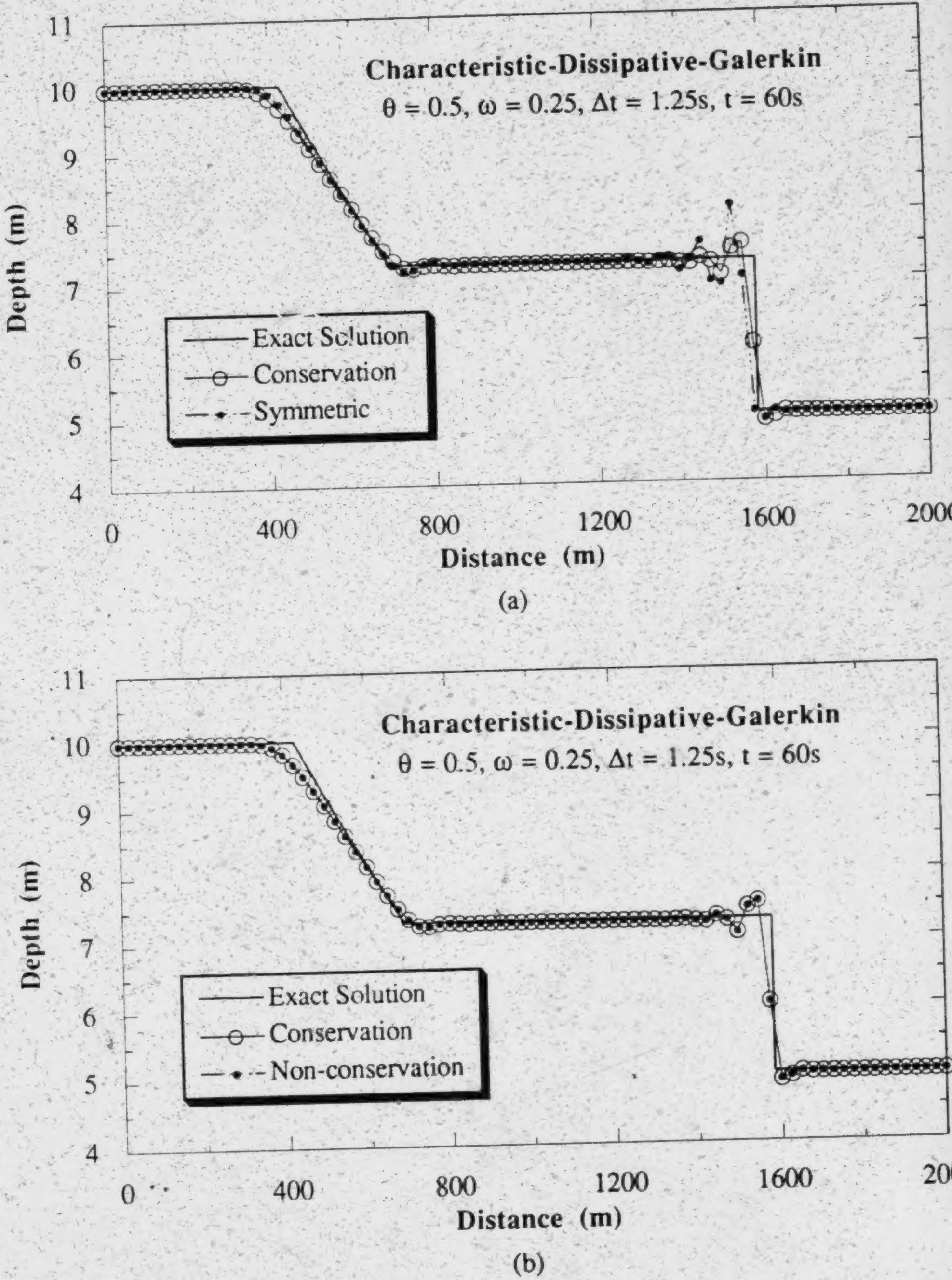


Figure 5.114 Non-linear perturbation (t = 14.0s), Characteristic-Dissipative-Galerkin method, **F** = 0.5, θ = 0.5, ω = 0.25, C = 1.0, Δt = 0.4317s
(a) conservation form of the equations vs. symmetric form; (b) conservation form of the equations vs. non-conservation form.



(a)

(b)

Figure 5.115 Dambreak, $H_1/H_0 = 0.5$, (t = 60s), Characteristic-Dissipative-Galerkin method, $\theta = 0.5$, $\omega = 0.25$, $\Delta t = 1.25$s (a) conservation form of the equations vs. symmetric form; (b) conservation form of the equations vs. non-conservation form.

6. HAY RIVER ANALYSIS

6.1 INTRODUCTION

6.1.1 General

To demonstrate the performance of the Characteristic-Dissipative-Galerkin (CDG) scheme further, it was applied to the determination of steady gradually varied flow profiles as well as the propagation of ice jam release surges over a 500 kilometer reach of the Hay River in Alberta/Northwest Territories. This test problem combined the complications of friction, varying channel width, and subcritical and supercritical flows. For steady flows, the ability of the model to step quickly to a converged solution was examined. In the unsteady case, the ice jam releases created a dambreak type of problem which began as a very dynamic situation then developed into an essentially kinematic wave problem as the disturbance propagated downstream.

Because it was intended to examine the applicability of the model to practical problems, a constant upwinding coefficient of $\omega = 0.25$ was used for both the steady and unsteady flow analysis. This limited the required input to physical parameters only.

6.1.2 Background

The Hay River originates in the foothills of the Rocky Mountains in northwestern Alberta and flows northeast to the Great Slave Lake in the Northwest Territories. Figure 6.1 shows a location sketch of the Hay River basin. The watershed encompasses approximately 48,000 square kilometres, extending into northeastern British Columbia as well. Details of the hydrologic aspects of the Hay River basin may be found in another study (Gerard and Stanley, 1988).

6.1.3 Geometric Data

The Town of Hay River is located at the mouth of the Hay River, at Great Slave Lake. Because the town has suffered severe spring flooding on numerous occasions, the river has been the object of a series of investigations into the formation and release of ice jams and their effects on water levels at the town (Gerard *et al*, 1988; 1990). As a result, geometric data extending over a reach of about 500 km (encompassing two waterfalls and numerous subcritical/supercritical flow transitions), was available for this investigation.

Water surface slopes and widths for the study reach, extending from the Highway 58 crossing of the Chinchaga River (Station 615 km) to the Town of Hay River (Station 1108 km), were determined based upon 1:50000 scale National Topographic Series (NTS) maps in conjunction with surveyed river cross sections at selected sites. An equivalent reach average rectangular section, based on the 1:2 year return period flood, was used to approximate the channel geometry at the survey sites. This facilitated the determination of an equivalent bed elevation, which was then extended throughout the entire domain using the water surface slopes obtained from the NTS maps. Figure 6.2 shows the resulting bed profile which was used as input to the model, as well as the survey sites; while Figure 6.3 illustrates the channel top widths obtained from the NTS maps together with those measured at survey sites. These top widths were used for the intermediate reaches between survey sites. Further details of the determination of these values are available in a separate document (Gerard *et al*, 1990). Because of the highly variable nature of these widths, this data was smoothed to obtain an alternate data set for the numerical model. The original and smoothed data sets allowed for examination of the effects of such rapid width variations on the model behavior and results.

The effective hydraulic roughness, k, was also determined on a reach averaged basis using the surveyed cross sections, water levels and the discharge on the day of the survey

(Gerard *et al*, 1990). Results varied from about 0.05 to 0.55 m. A review of this information and the circumstances at each survey site indicated a constant value of k = 0.2 m would be reasionable for use throughout the entire domain for this investigation.

6.1.4 Discretization

Throughout the upstream portion of the study reach a section spacing of 1 km was used. This spacing was consistently decreased (15% decrements) to 200 m just upstream of the steep reach in the vicinity of the falls (between stations 1027 and 1032.5 km) and then increased again to 500 m further downstream (station 1045 km). A total of 617 cross sections were used in the analysis.

6.1.5 Boundary and Initial Conditions

For the purpose of this investigation, lateral inflows along the study reach were neglected. Therefore, a constant (carrier) discharge was specified as the initial condition for both steady and unsteady flow runs. For convenience, this was based on the 1:2 year flood used in determining the bed profile. As this discharge varied from about 300 to 700 cms over the study reach (Gerard *et al*, 1990), an average value of 500 cms was taken for the analysis.

The initial depths for the steady flow runs were based on the estimated mean bed profile, widths and hydraulic roughness. For the unsteady flow simulation, the gradually varied flow profiles obtained from the steady flow analysis were used to establish the initial conditions.

Although the study domain encompassed a number of subcritical and supercritical reaches, the flow remained subcritical at both the upstream and downstream extremities. Therefore, one upstream and one downstream boundary condition were required for the simulation. The upstream boundary condition was taken as the carrier discharge, while at the downstream end a constant water level of 156.9 m, representing a water level on Great Slave Lake, was used.

6.2 STEADY FLOW ANALYSIS

As stated above, this analysis was conducted for a discharge of 500 cms, and was run for two sets of data. A number of trial runs were first conducted to determine the maximum allowable initial time step increment. This was obtained by starting with a Δt of 1,000,000 s and then reducing it by a factor of 10 until a stable solution was achieved. All runs were fully-implicit (θ=1.0) with a coarse convergence tolerance of 0.1.

For the data file with the unsmoothed channel widths, the time step was initially taken as 10 seconds and was increased by a factor of 10 every 10 time steps until a steady state (converged) solution was achieved, after 50 time steps. Figure 6.4 illustrates the steady flow profile obtained for the entire reach. To examine these results more closely, a section in the vicinity of the steep falls is expanded in Figure 6.5.

For the smoothed width data file, the time step could initially be taken as 100,000 s and a steady state (converged) solution was achieved after only 10 time steps. Figure 6.6 illustrates the steady flow profile obtained for the entire reach, while the expanded section is shown in Figure 6.7.

The two data files converged to a steady state solution after approximately $t = 1\mathrm{x}10^6$ s, although the smoothed data was able to achieve this in 10 rather than 50 time steps. The ultimate water surface profiles for the two runs were within about 0.1 to 0.2 m of each other, on average, with larger variations (0.3 to 0.5 m) in the vicinity of Station 1050 km. This is an area of rapidly changing slopes and widths, as illustrated in Figures 6.2 and 6.3. The solution for the smoothed data converged to a steady discharge within

0.05% of 500 cms throughout the domain, except in the vicinity of rapid slope t where localized fluctuations in the order of 0.3% were observed. An extreme d (25%) was observed in the vicinity of the falls. This would be expected based results of the hydraulic jump tests discussed in Chapter 5. However, this was co a localized effect and did not destabilize the solution. Results were not as good for the NTS map (unsmoothed) data where discharge varied 1 to 2 % on average, and up to 7% in the vicinity of rapid slope transitions. The extreme was, however, limited to 25% (in the same location as for the smoothed data).

6.3 UNSTEADY FLOW ANALYSIS

6.3.1 Introduction

As stated above, the results of the steady flow analysis were taken as the initial condition for the unsteady flow simulations. An ice jam was then superimposed on the flow at a selected location to act, essentially, as a surge generator. The first location chosen was roughly 190 km upstream of the town at Station 920 km. The second jam location was approximately 13 km upstream of the town, at Station 1095.6 km (W.S.C. Gauge site). All runs were semi-implicit ($\theta=0.5$) with a more restrictive convergence tolerance of 10^{-5}.

6.3.2 Assumptions for Ice Jam Simulation

As discussed in the Hay River study (Gerard, Hicks and Jasek, 1990) a number of simplifying assumptions concerning the initial jam shape and failure mechanism were used in conducting the unsteady flow analysis. For example, as illustrated in Figure 6.8, the ice jam was assumed to have a toe length equal to the grid spacing used for the geometric data. The gradually varied (M1) flow profile upstream of the jam was approximated by projecting a horizontal line back from the jam to an intersection with the upstream water level. The flow depth to the phreatic surface within the jam was determined based on an equilibrium section of a fully developed jam, using the following relationship developed from the approach suggested by Beltaos (1980):

$$\eta = 0.38\,\zeta + \frac{5.75}{\mu}\left\{1+\sqrt{1+0.07\,\mu\,\zeta\left(\frac{k_i}{k}\right)^{1/4}}\right\} \qquad [6.1]$$

where

$$\eta \equiv \frac{d}{SB} \qquad [6.2]$$

and

$$\zeta \equiv \frac{\left(qk^{1/6}\big/\sqrt{gS}\right)^{3/5}}{SB} \qquad [6.3]$$

in which μ is an internal friction - porosity coefficient (taken as 1.0 for this investigation), d is the depth to the phreatic surface, S the slope of the energy line, B the pack width, and q the discharge per unit width of channel. k is the composite roughness, based on the ice roughness, k_i and the bed roughness, k_b, such that:

$$k = \left(\frac{k_i^4 + k_b^4}{2}\right)^{1/4} \qquad [6.4]$$

A constant value of 2 m was used for the hydraulic roughness of the underside of the ice pack for all runs conducted in this analysis (Gerard *et al*, 1990). Again, for the bed, a constant value of 0.2 m was used throughout the study reach.

The release of the ice jam was assumed to be instantaneous along its length, resulting in a uniform acceleration of the stored water over its entire length. In addition, it was assumed that the ice in the pack followed the water and therefore had no resistance effect on the flow after the time of release. Finally, it was assumed that the entire reach downstream of the toe of the jam was open (no ice cover).

6.3.3 Ice Jam Surge Releases at Station 920 km

This ice jam release location was selected for two reasons. First, it allowed a sufficient distance for the initially dynamic surge to transform into an essentially kinematic wave. Second, it would require the flood wave to pass through the steep reach (Alexandra and Louise Falls) upstream of the town. Finally it allowed room upstream for a very long (50 km) jam, in order to effect a large increase in water levels and discharges at the town.

Figure 6.9 illustrates the variation in width over the reach containing the jam, while Figure 6.10 shows the initial discharge, determined from the calculated steady flow situation. The difference between the widths and the initial discharge conditions for the two data sets lead to a slightly different jam profile, as seen in Figure 6.11. The jam itself was 50 km long with the backwater profile extending a further 40 km upstream, approximately. The model was run using a $\Delta t = 60$ s, which corresponds to a Courant number ranging from approximately 0.1 to 0.5 for the element lengths of 1000 and 200 m, respectively. The Courant number was determined based upon the time of travel of the flood peak from the toe of the jam to the town.

Figure 6.12 illustrates the hydrographs at Stations 950, 1031 and 1108 km (at the town of Hay River) using the varied width data. Results for the smoothed data are shown in Figure 6.13. Although the results for the two runs were nearly identical just downstream of the jam at Station 950 km, the carrier discharge was somewhat above 500 cms for the unsmoothed data. In general, the smoothed data file required fewer iterations per time step (2 - 3 compared to 3 - 4). As seen in the figures, the peaks arrived at the downstream locations at nearly identical times, although the magnitude was about 2 to 3% lower for the varied width data set. The average propagation speed of about 1.8 m/s (1.2 miles per hour) is consistent with field observations (Gerard, Hicks and Jasek, 1990). It is interesting to note that there is very little attenuation of the peak between Station 1031 km and the town, indicating that the wave was nearly kinematic by Station 1031 km.

Figure 6.14 shows the loop rating curves obtained at Station 950 km, just downstream of the toe of the jam, for both data sets. In contrast, the rating curves shown in Figure 6.15 for Station 1031 km (just upstream of the falls) indicate that the wave was approaching a kinematic condition, as almost no looping was observed. The trend continued as seen in Figure 6.16, which shows the rating curves at the town of Hay River (station 1108 km).

6.3.4 Ice Jam Surge Releases at Station 1095.6 km

The second jam location was approximately 12 km upstream of the town, at Station 1095.6 km (W.S.C. Gauge). This allowed for the investigation of a very dynamic event at the town in contrast to the previous test.

Figure 6.17 illustrates the variation in width over the reach containing the jam, while Figure 6.18 shows the initial discharge, based on the steady flow analysis. As seen in Figure 6.11, the jam was 20 km long with the backwater profile extending only about 5 km further upstream. Because the differences between the widths and the initial discharge conditions for the two data sets were less substantial than at the upstream site, the jam profiles were much more similar. The model was again run using a $\Delta t = 60$ s corresponding to a Courant number ranging from approximately 0.1 to 0.6 for the element lengths of 1000 and 200 m, respectively.

Figure 6.20 illustrates the hydrographs at Stations 1098 and 1108 km using the NTS map data, while those for the smoothed data are shown in Figure 6.21. Again, the results for the two runs were nearly identical with the smoothed data file requiring fewer iterations per time step. The fluctuations observed on the upstream hydrographs occurred due to the various wave component generated out of the different parts of the released jam. This was due to the varying depth and constant discharge conditions specified just prior to jam release. These multiple waves eventually combined to form a single smooth wave, as seen in the hydrographs for Station 1108 km at the town.

Figure 6.22 shows the loop rating curves obtained at Station 1098 km, just downstream of the toe of the jam, for both data sets. The fluctuations on the rising limb are associated with the multiple peaks observed in the hydrographs. The rating curves shown in Figure 6.23, for the town of Hay River (station 1108 km), indicate that the wave was already much less dynamic, as significantly less looping was observed. Again, results were quite similar for the two sets of data.

6.3.5 Discussion of Results

Both sets of tests illustrate the applicability of the CDG method to practical problems, particularly given the extreme variations in width and bed slope over the study reach. The fact that the model was able to establish steady flow profiles through numerous supercritical/subcritical transitions including the two sets of falls, is unique in itself. However, the results of these tests are particularly significant in terms of the unsteady flow analysis as the CDG scheme provided stable solutions not only for the extremes of dynamic and kinematic wave conditions, but through the transition between the two as well, even though the upwinding is based entirely on the dynamic terms in the equations. The fact that stable solutions were obtained, for both steady and unsteady flow, without adjusting the upwinding parameter means that the practical application of the model would only require calibration of physical parameters (ie. roughness).

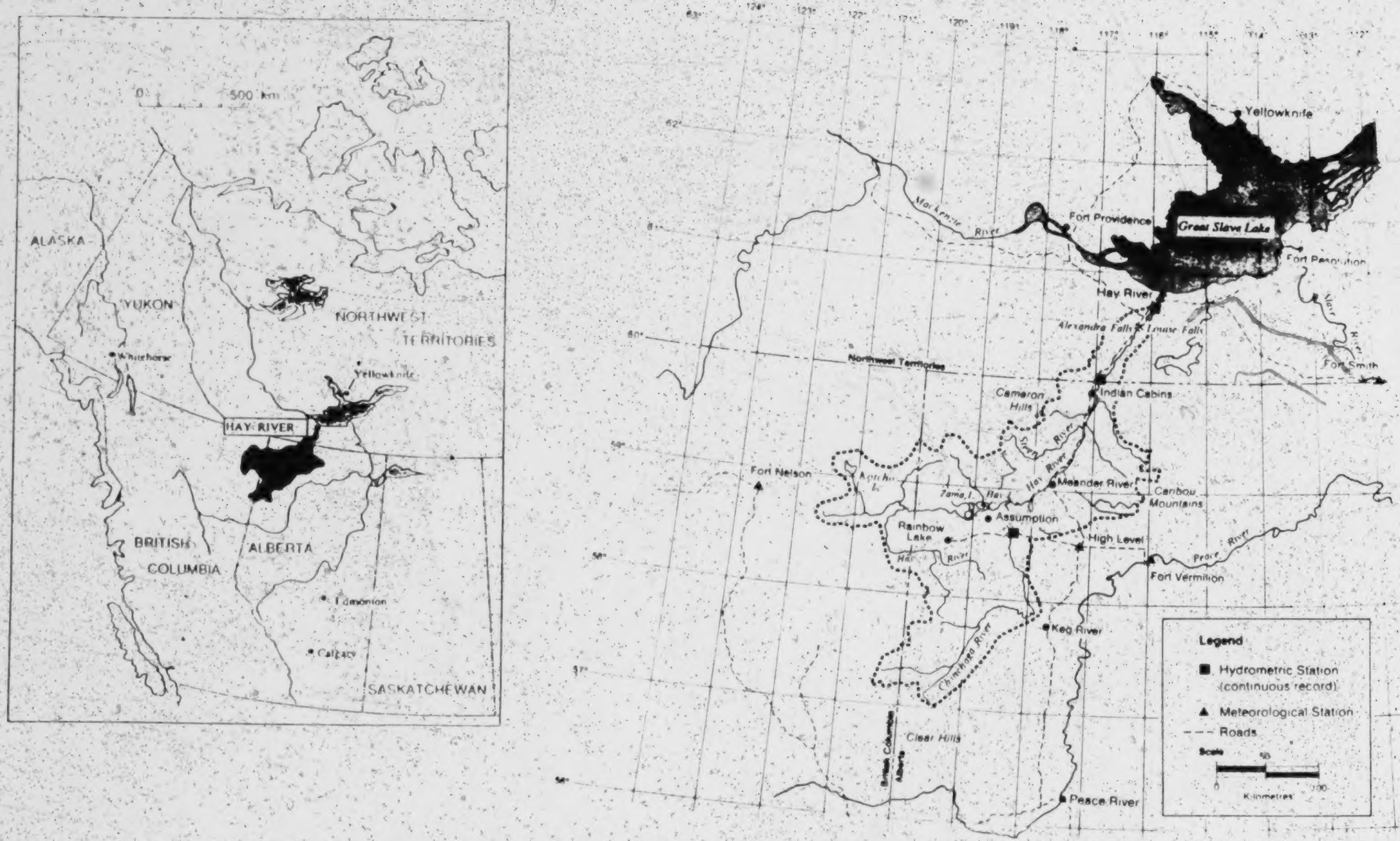

0
500 km
ALASKA
YUKON
NORTHWEST
TERRITORIES
Whitehorse
Yellowknife
HAY RIVER
BRITISH
COLUMBIA
ALBERTA
Edmonton
Calgary
SASKATCHEWAN
Yellowknife
Great Slave Lake
Fort Providence
Fort Resolution
Mackenzie River
Hay River
Alexandra Falls
Louise Falls
Slave River
Fort Smith
Northwest Territories
Cameron Hills
Indian Cabins
Steen River
Hay River
Fort Nelson
Meander River
Caribou Mountains
Zama L.
Assumption
Rainbow Lake
Hay River
High Level
Peace River
Fort Vermilion
Keg River
Chinchaga River
Clear Hills
British Columbia
Alberta
Peace River
Legend
■ Hydrometric Station (continuous record)
▲ Meteorological Station
Roads
Scale
0
50
100
Kilometres

Figure 6.1 Location sketch for the Hay River basin.

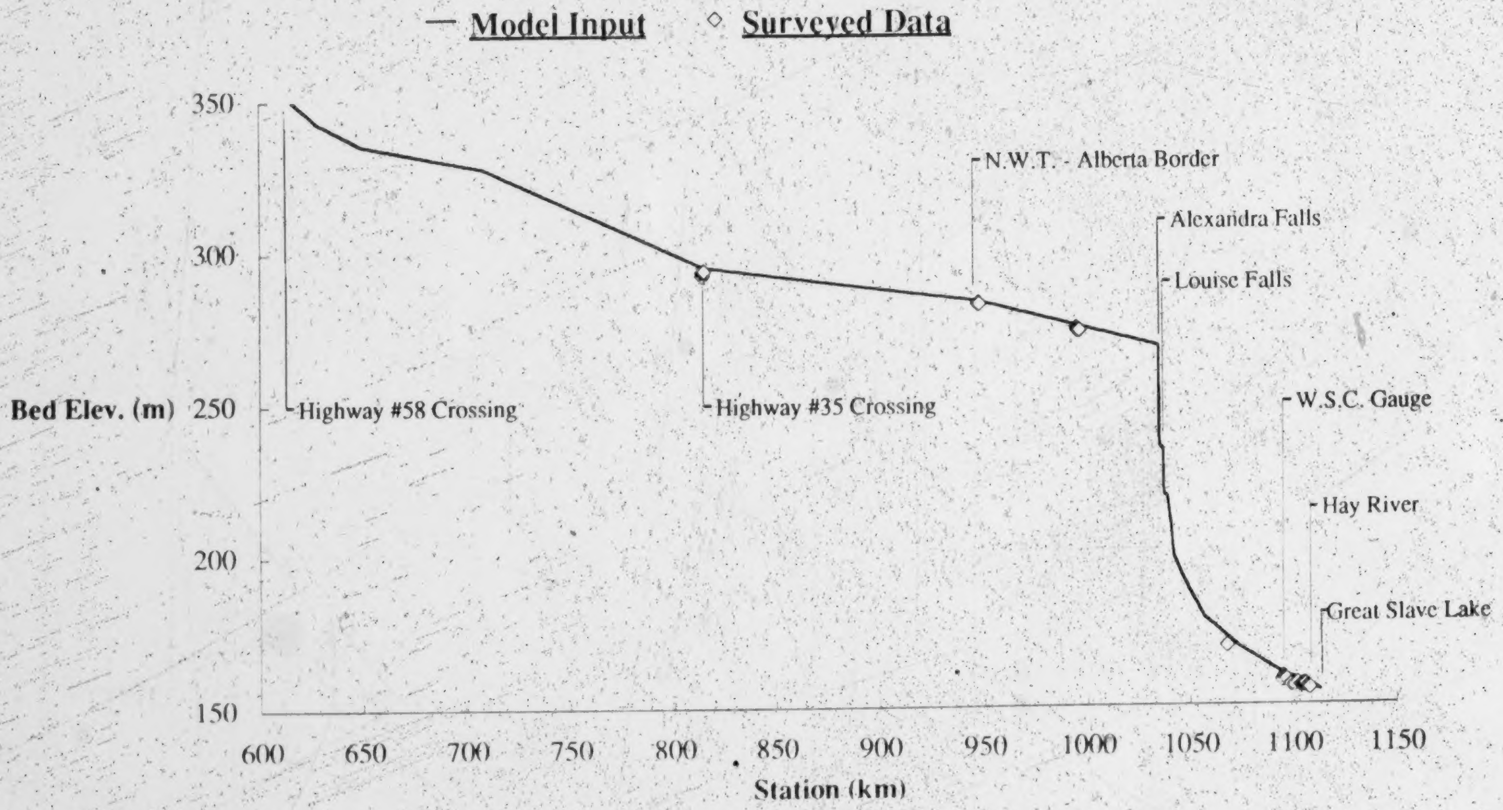


Figure 6.2 Bed profile of the Hay River.

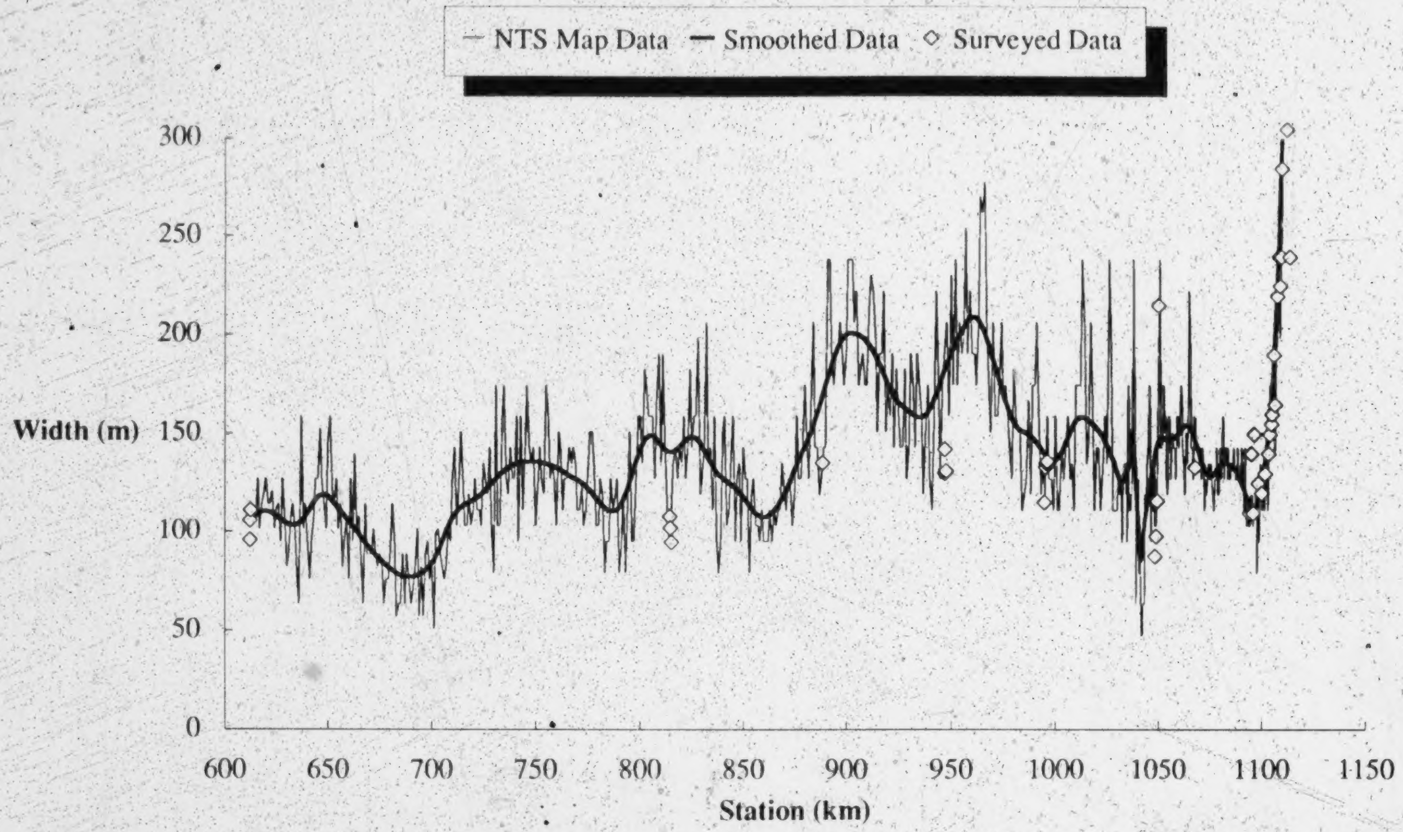


Figure 6.3 Hay River channel widths.

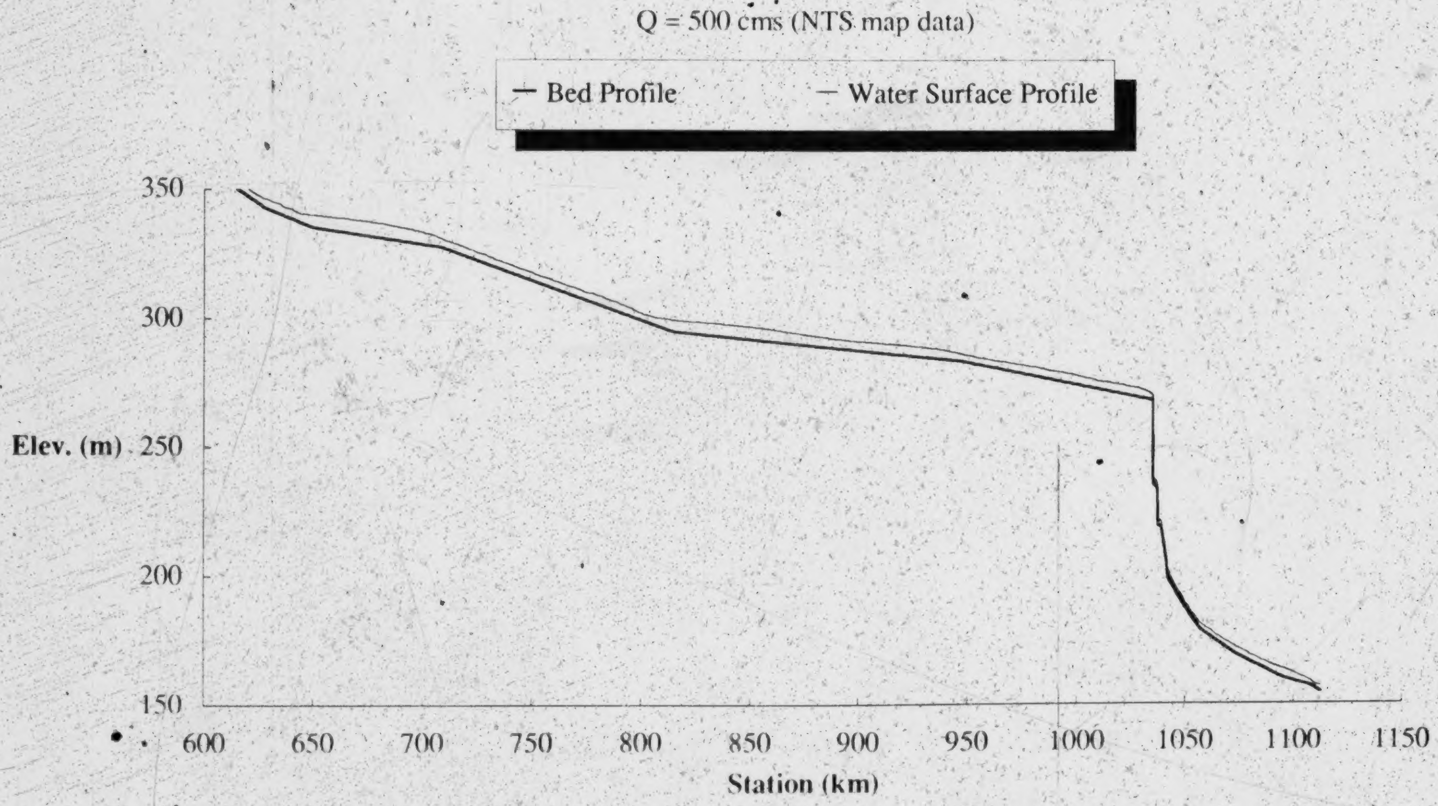


Figure 6.4 Steady gradually varied flow profile for the entire study reach, (NTS map data).

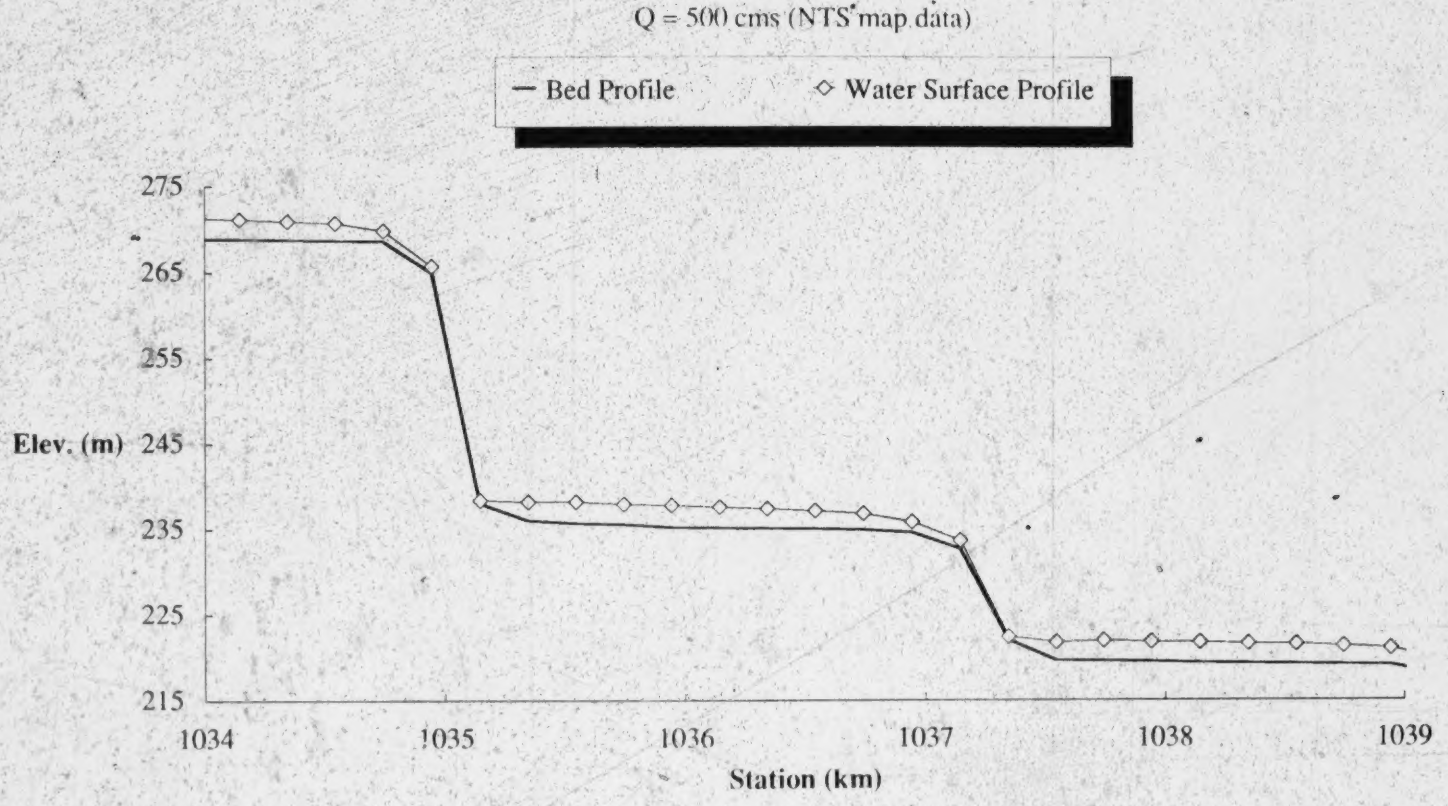


Figure 6.5 Steady gradually varied flow profile in the vicinity of the falls, (NTS map data).

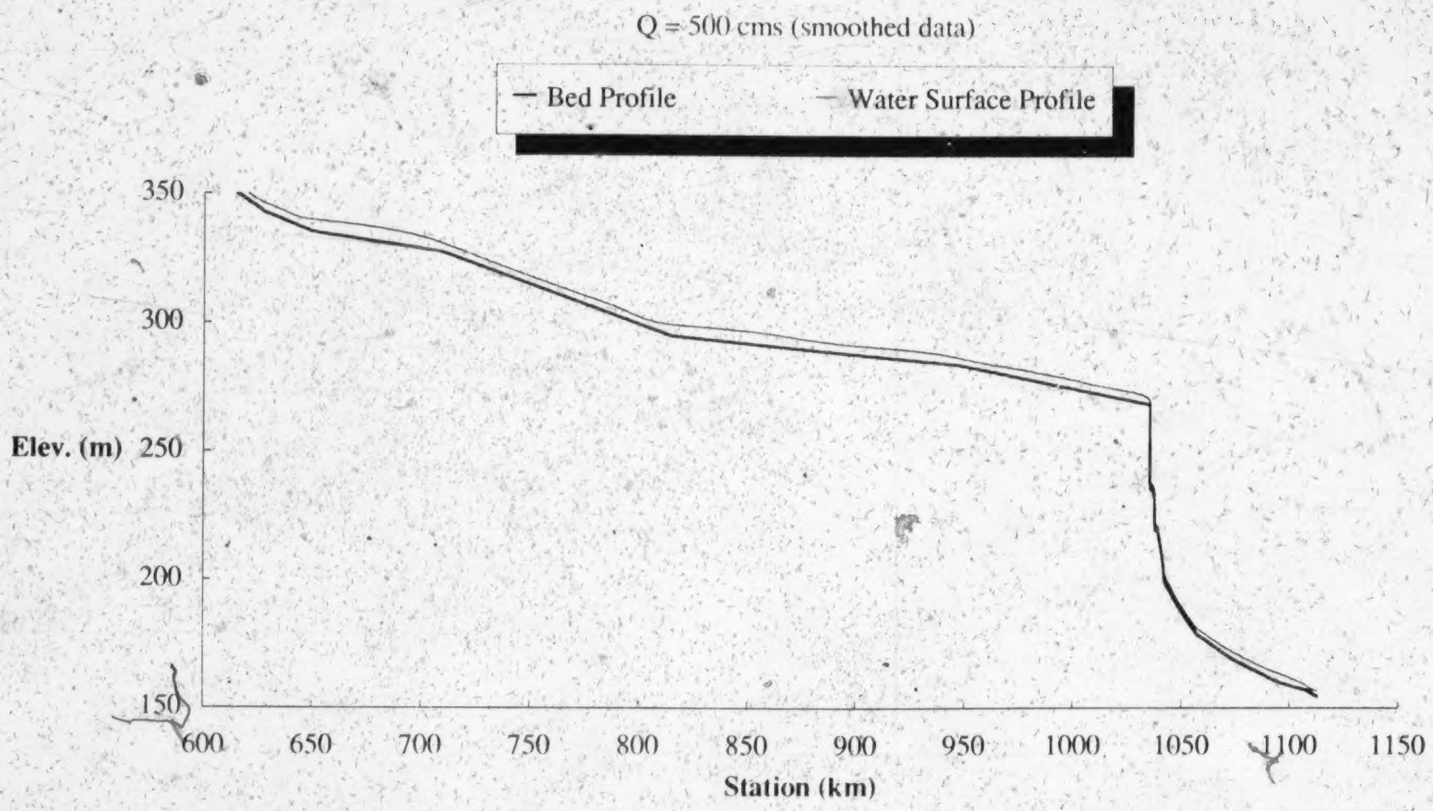


Figure 6.6 Steady gradually varied flow profile for the entire study reach, (smoothed data).

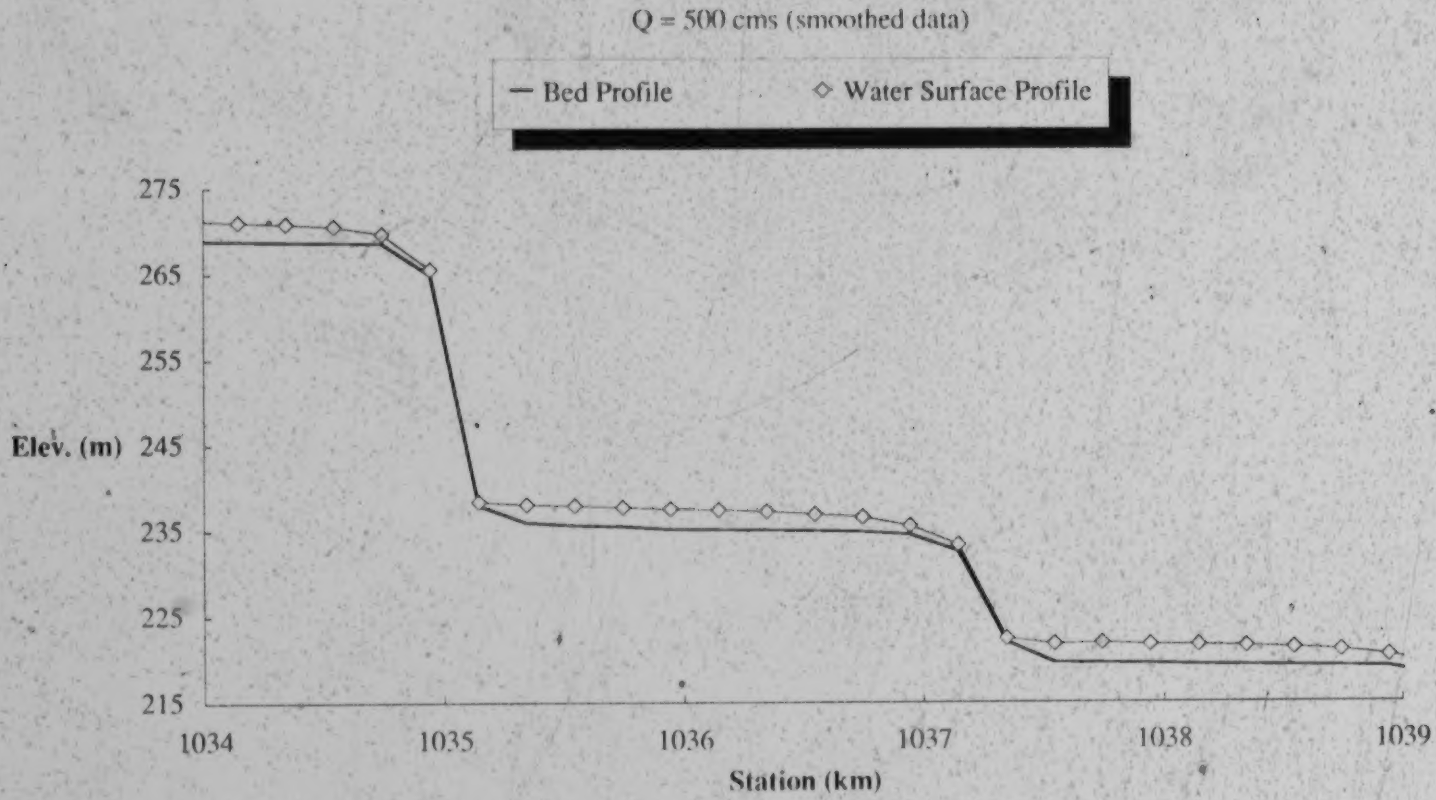


Figure 6.7 Steady gradually varied flow profile in the vicinity of the falls, (smoothed data).

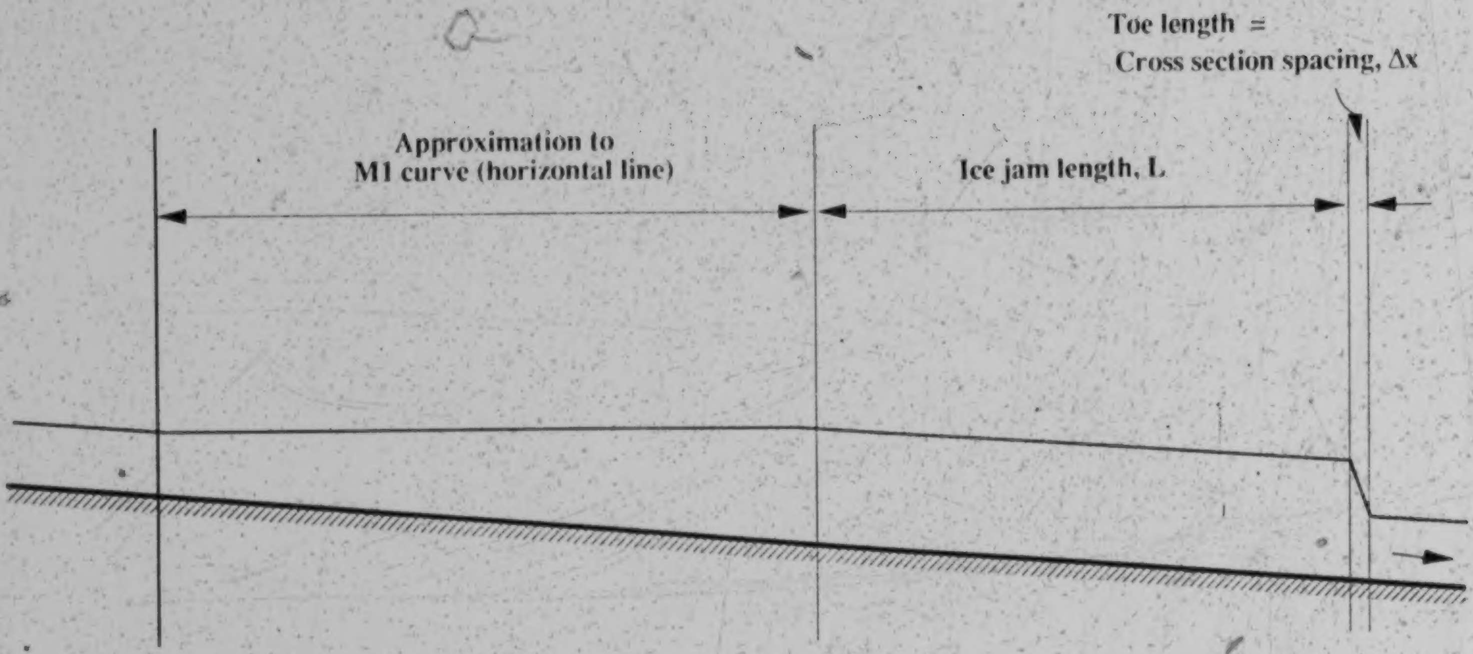

Toe length =
Cross section spacing, Δx
Approximation to
M1 curve (horizontal line)
Ice jam length, L

Figure 6.8 Schematic describing ice jam prior to failure.

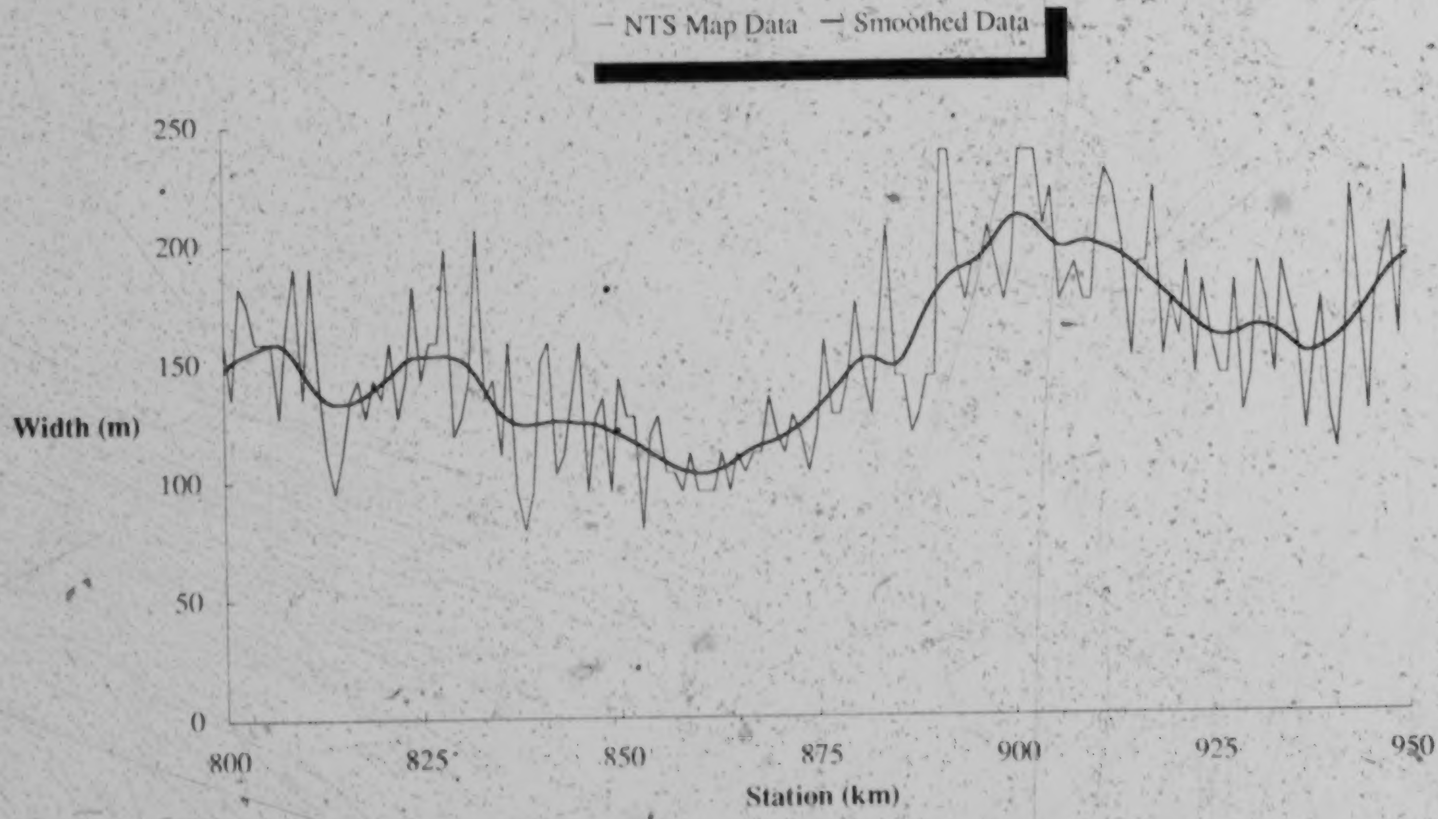


Figure 6.9 Variation in channel width over the ice jam reach (ice jam located at Station 920 km).

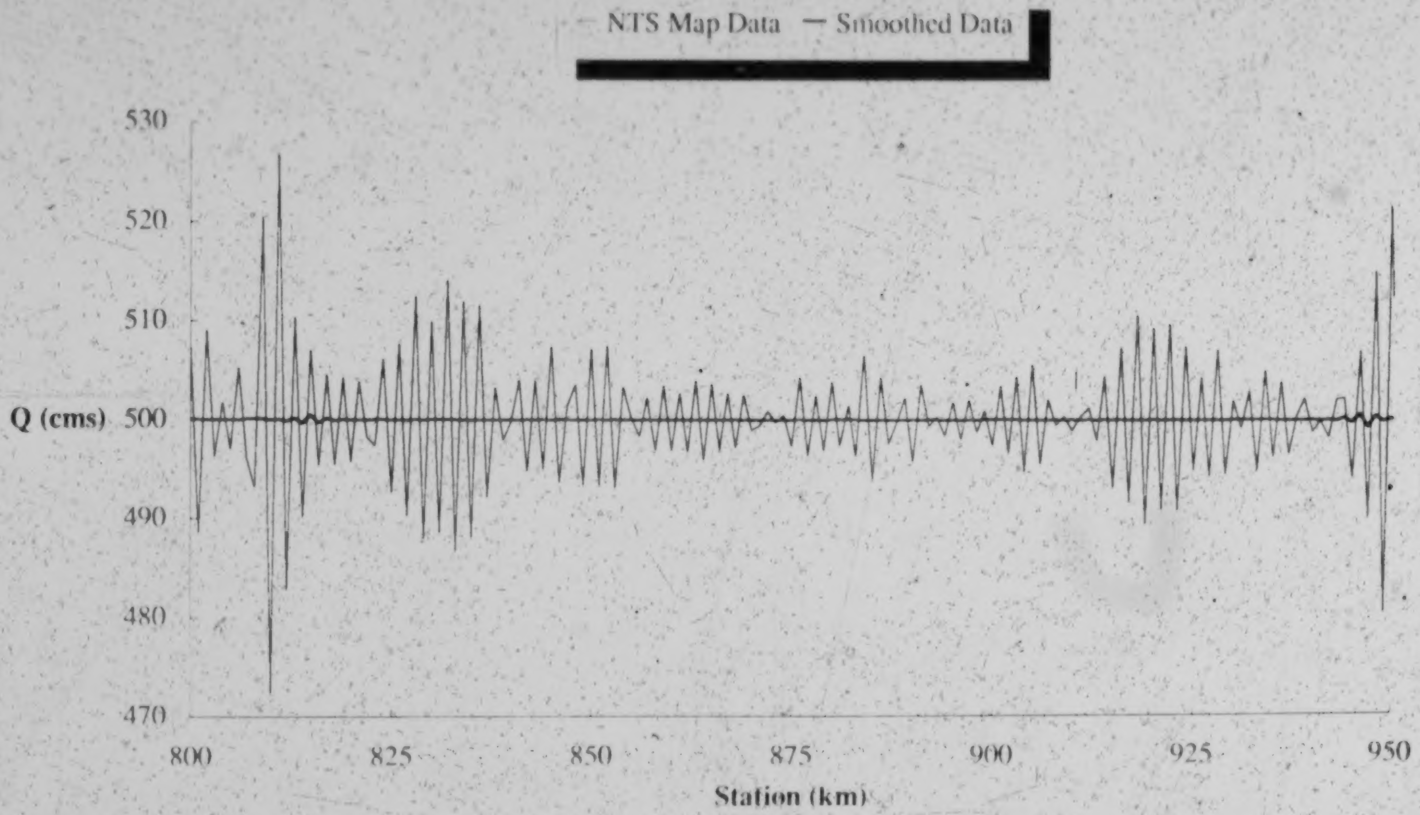


Figure 6.10 Variation in the initial discharge over the ice jam reach (ice jam located at Station 920 km).

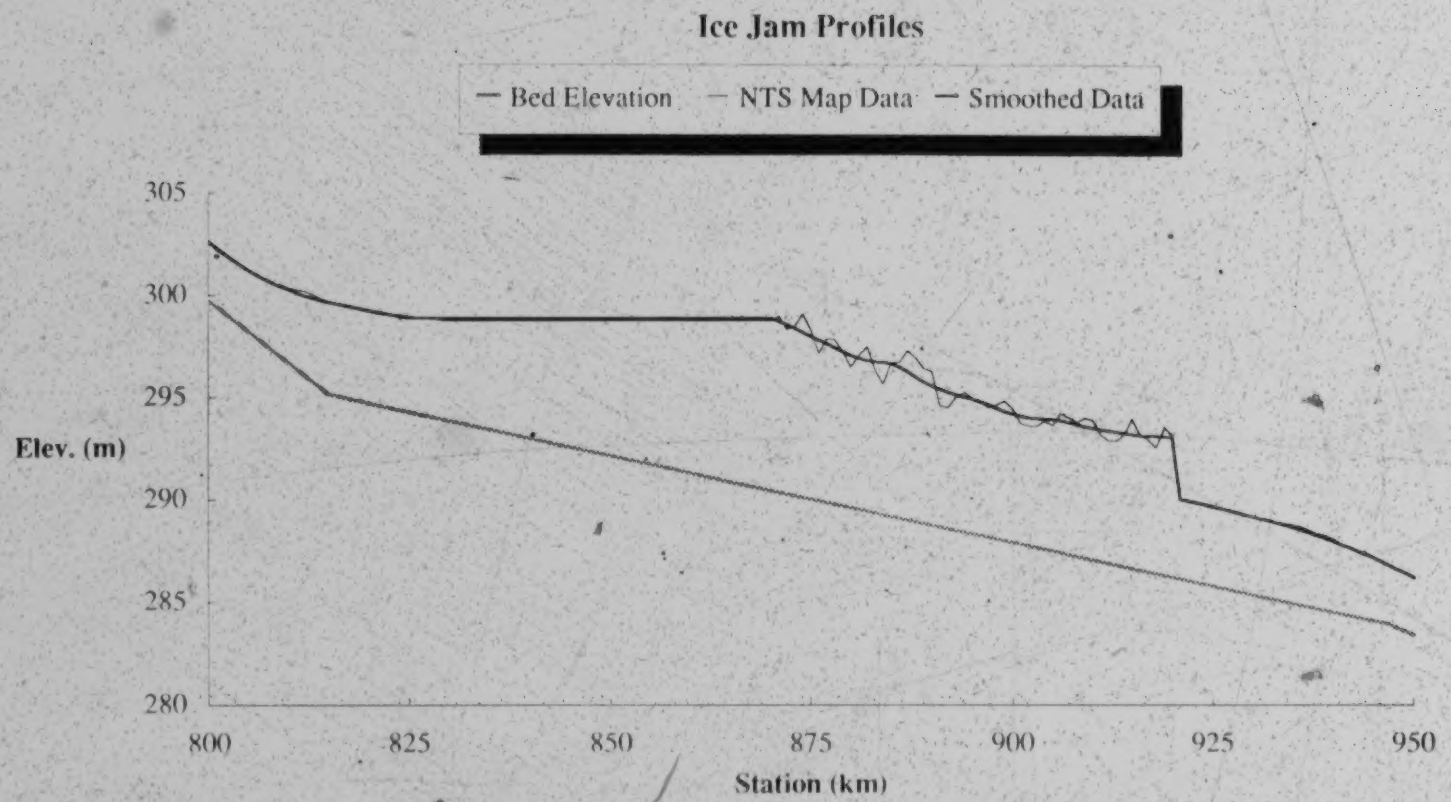


Figure 6.11 Ice jam profiles for jam located at Station 920 km.

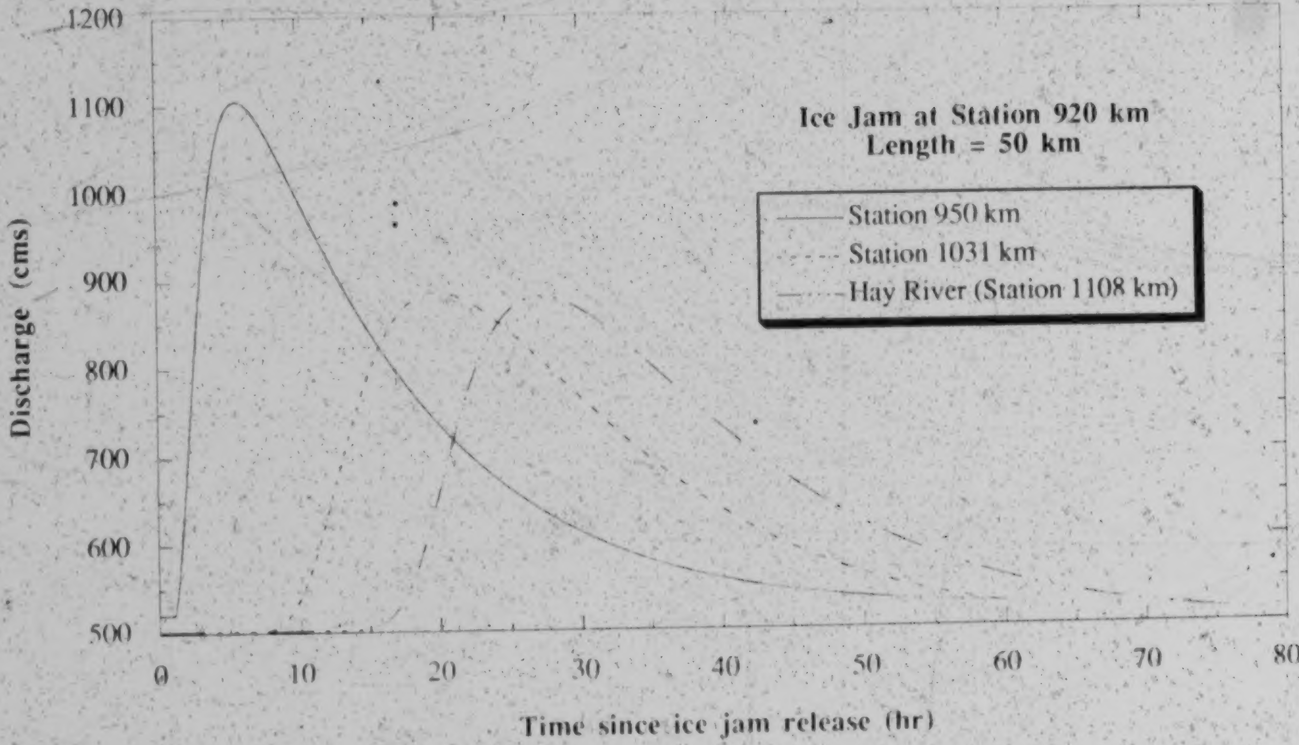


Figure 6.12 Hydrographs at Stations 950, 1031 and 1108 km for the jam released at Station 920 km (NTS map data).

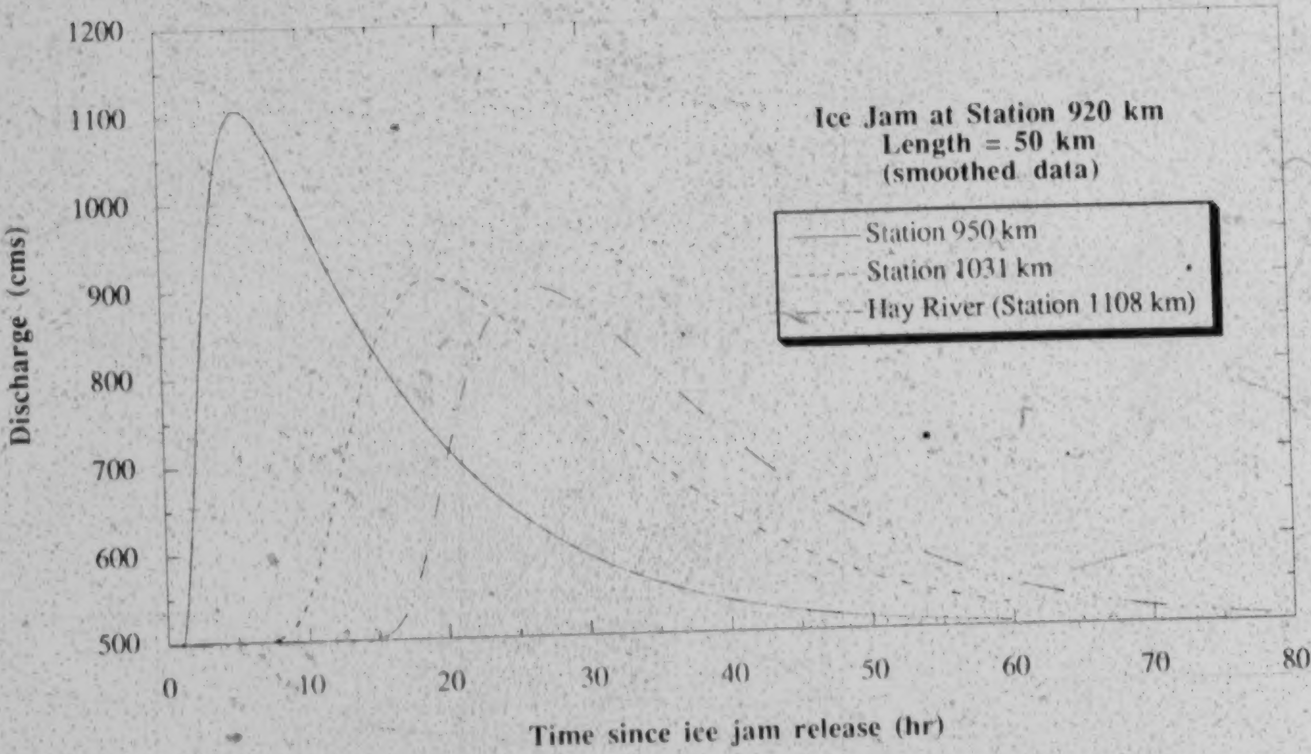
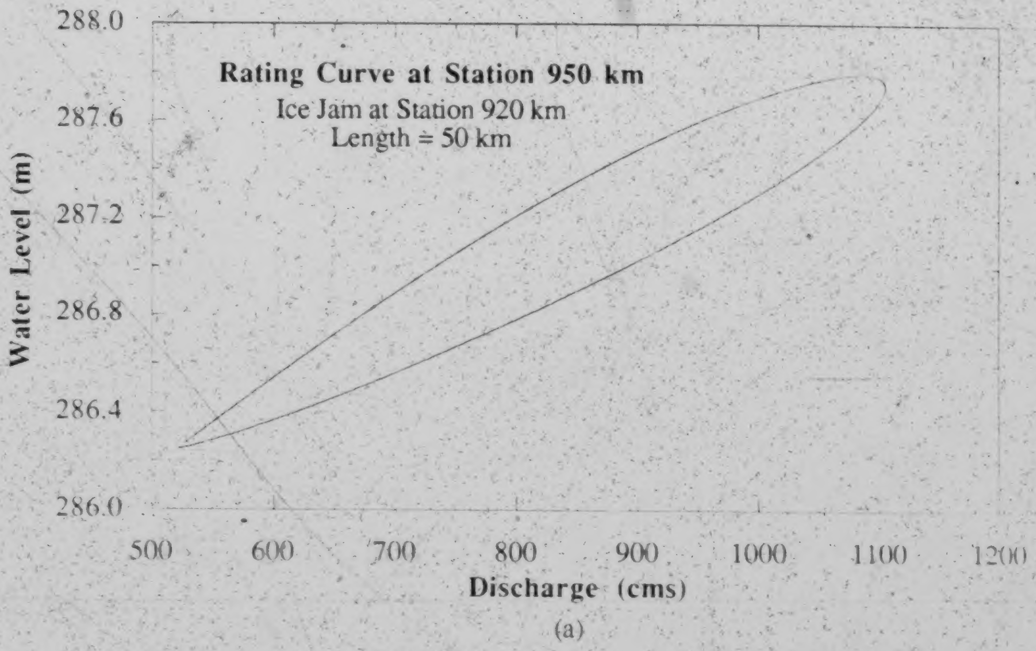


Figure 6.13 Hydrographs at Stations 950, 1031 and 1108 km for the jam released at Station 920 km (smoothed data).



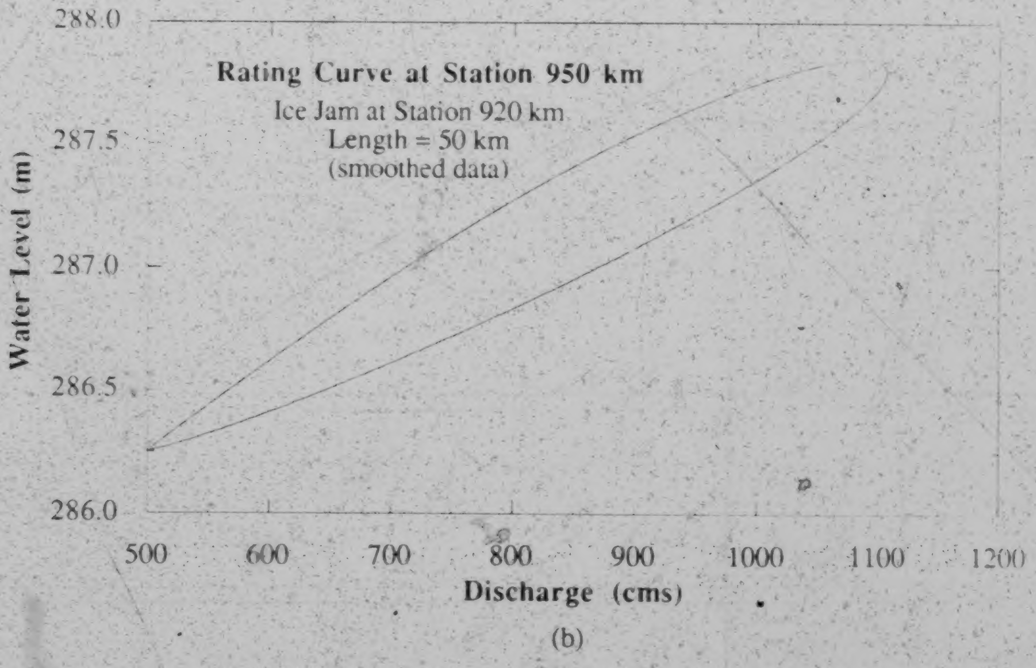


Figure 6.14 Rating curves at Station 950 km for the jam released at Station 920 km, (a) NTS map data, (b) smoothed data.

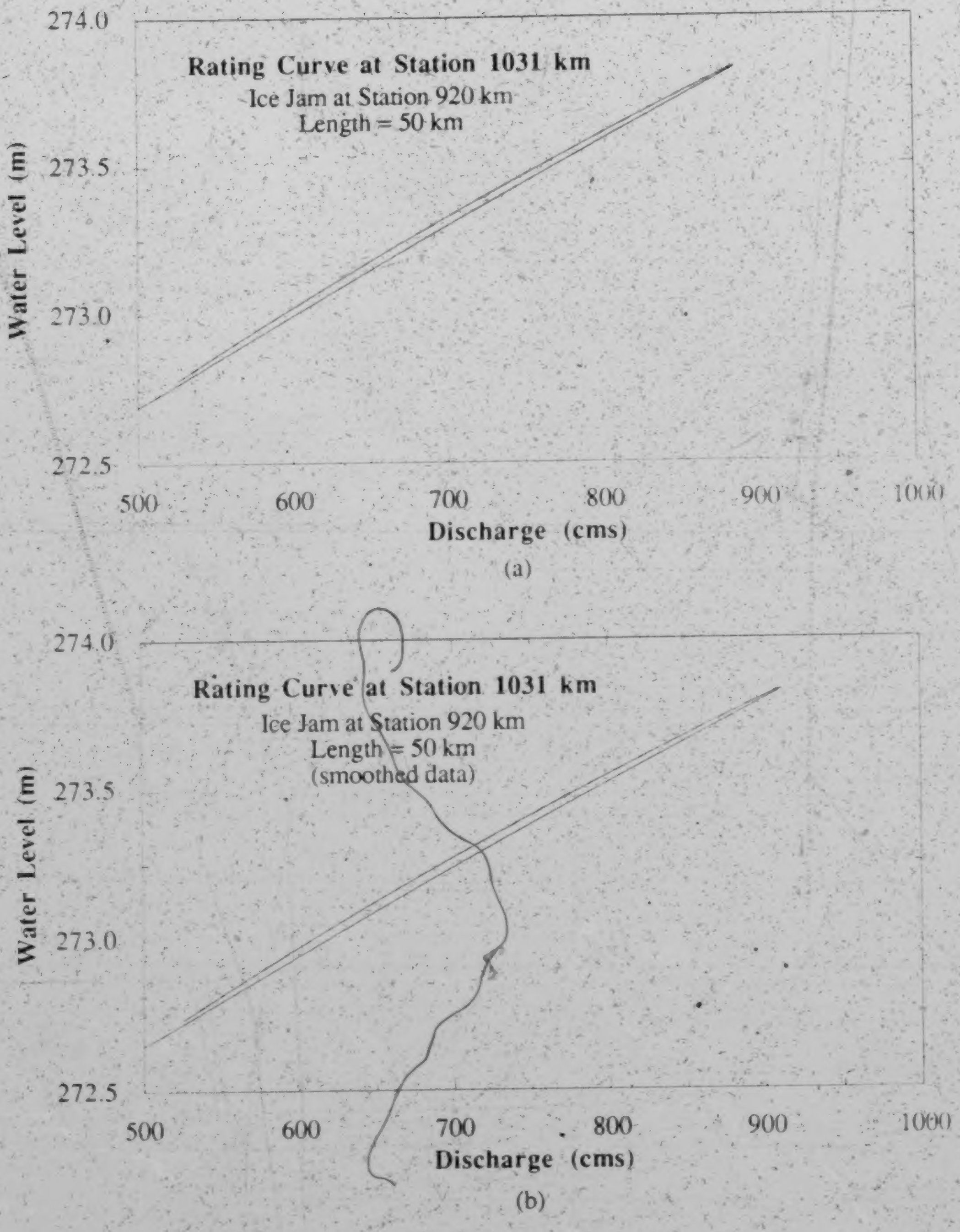
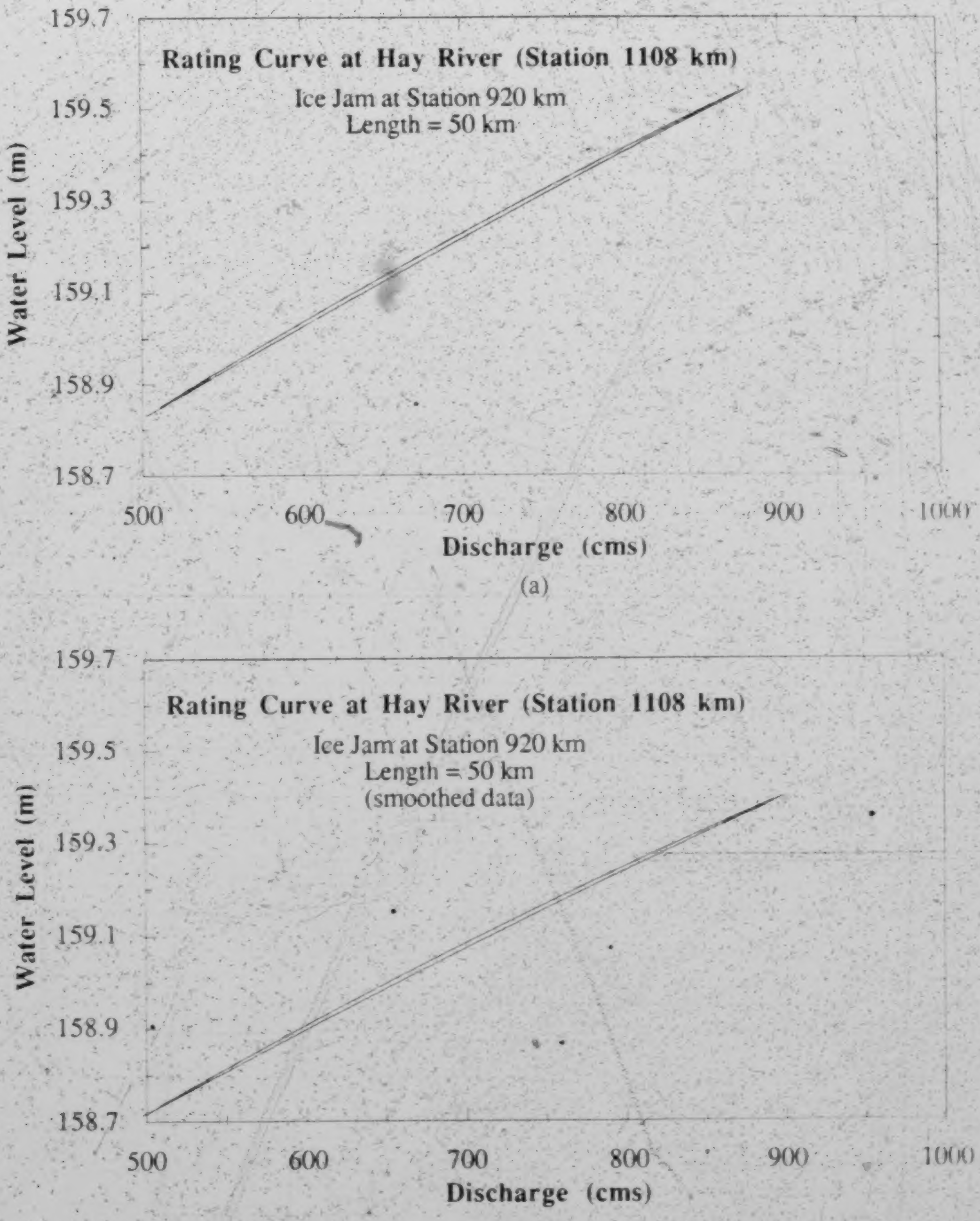


Figure 6.15 Rating curves at Station 1031 km for the jam released at Station 920 km, (a) NTS map data, (b) smoothed data.



(a)

(b)

Figure 6.16 Rating curves at Hay River for the jam released at Station 920 km. (a) NTS map data, (b) smoothed data.

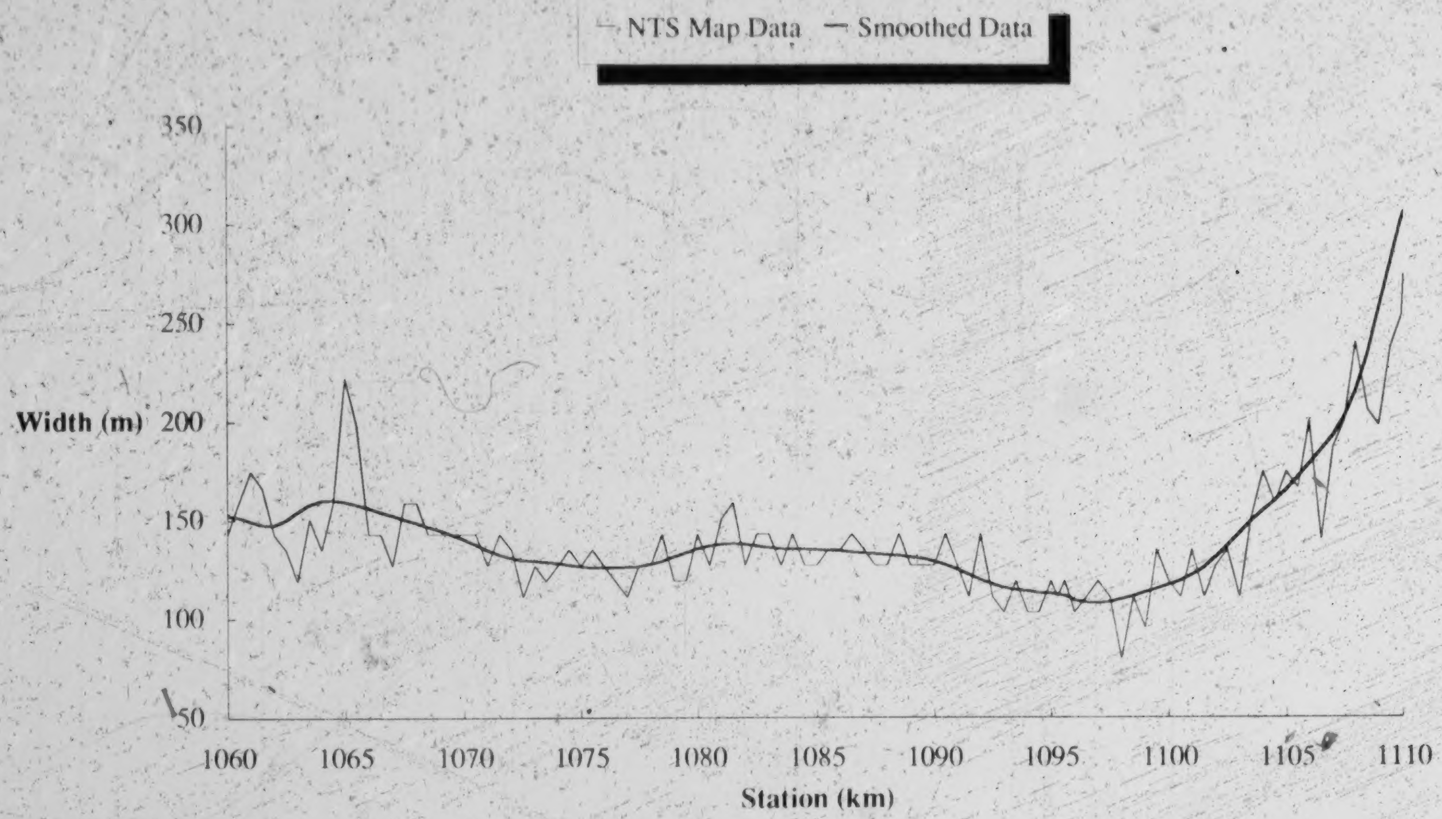


Figure 6.17 Variation in channel width over the ice jam reach (ice jam located at Station 1095.6 km).

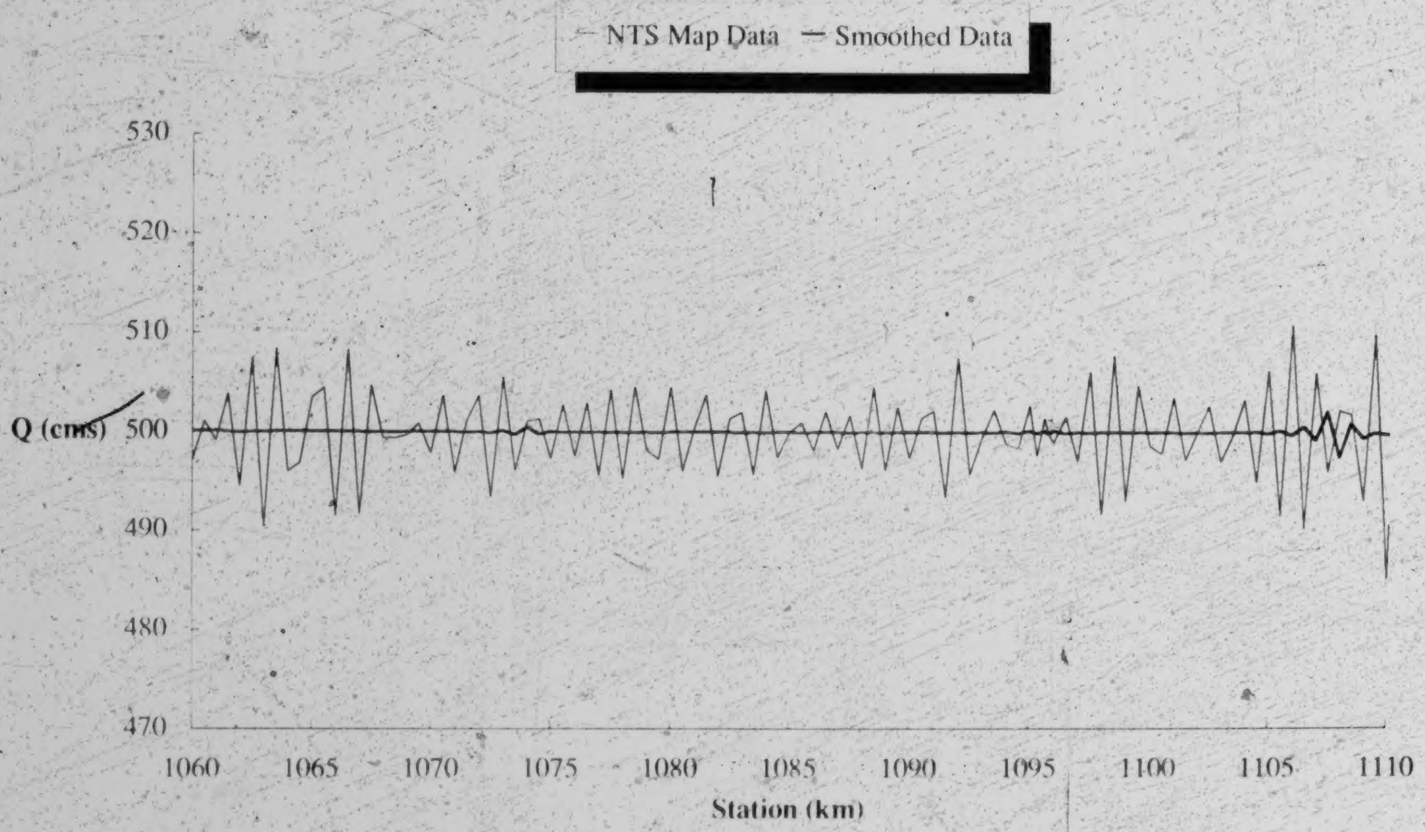


Figure 6.18 Variation in the initial discharge over the ice jam reach (ice jam located at Station 1095.6 km).

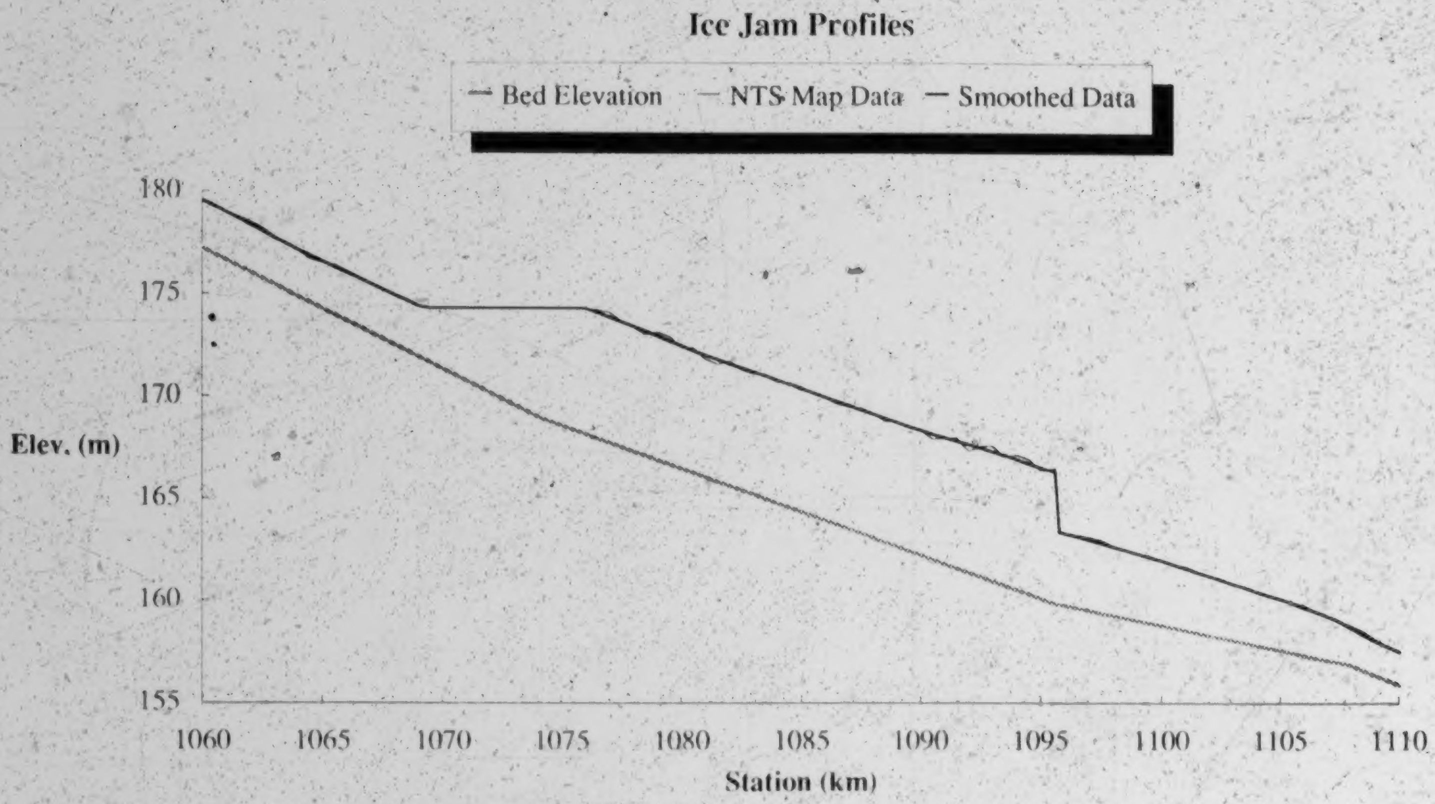


Figure 6.19 Ice jam profiles for jam located at Station 1095.6 km.

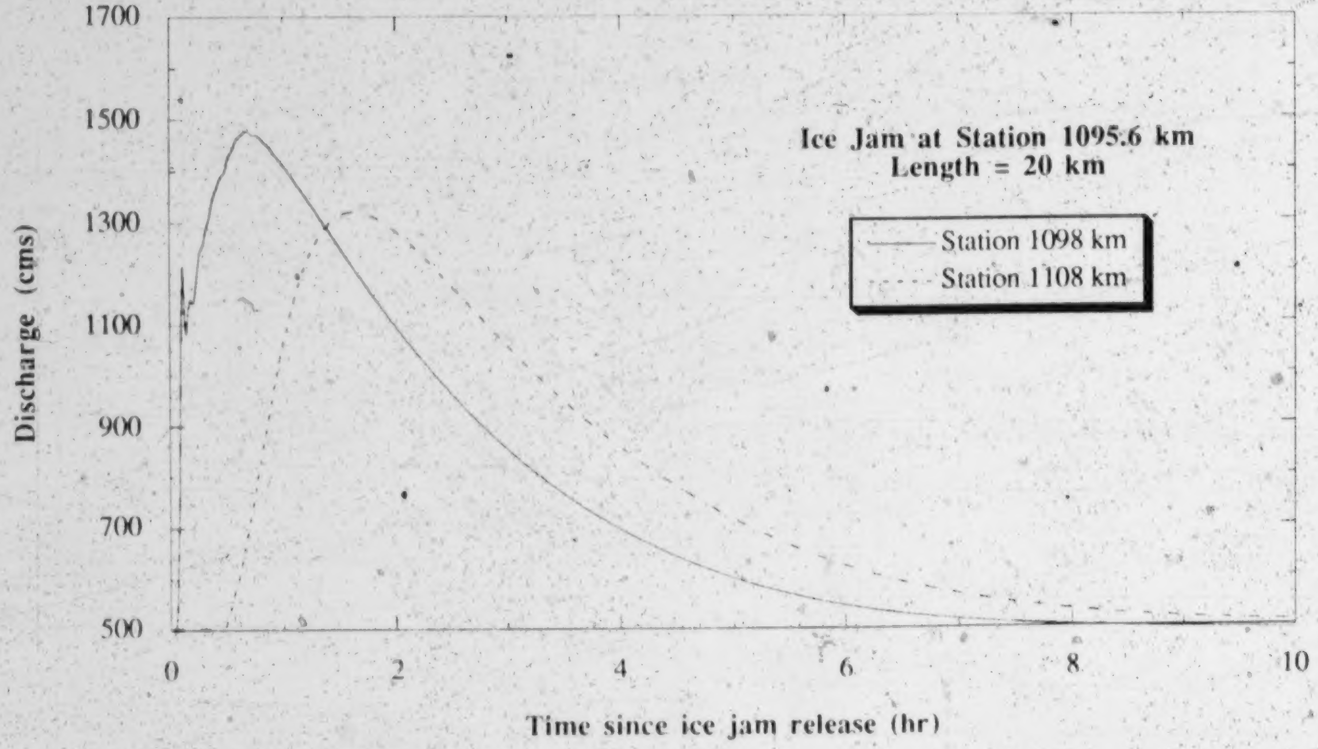


Figure 6.20 Hydrographs at Stations 1098 and 1108 km for the jam released at Station 1095.6 km (NTS map data).

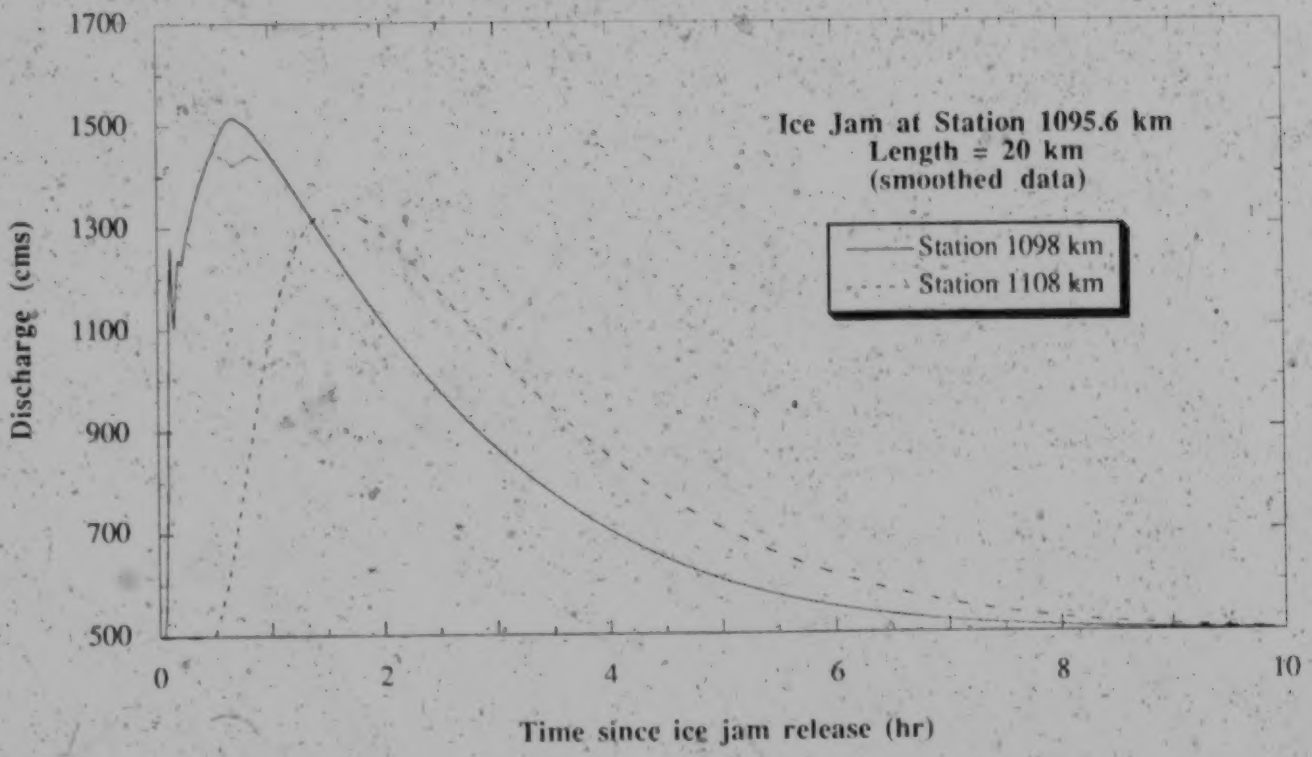


Figure 6.21 Hydrographs at Stations 1098 and 1108 km for the jam released at Station 1095.6 km (smoothed data).

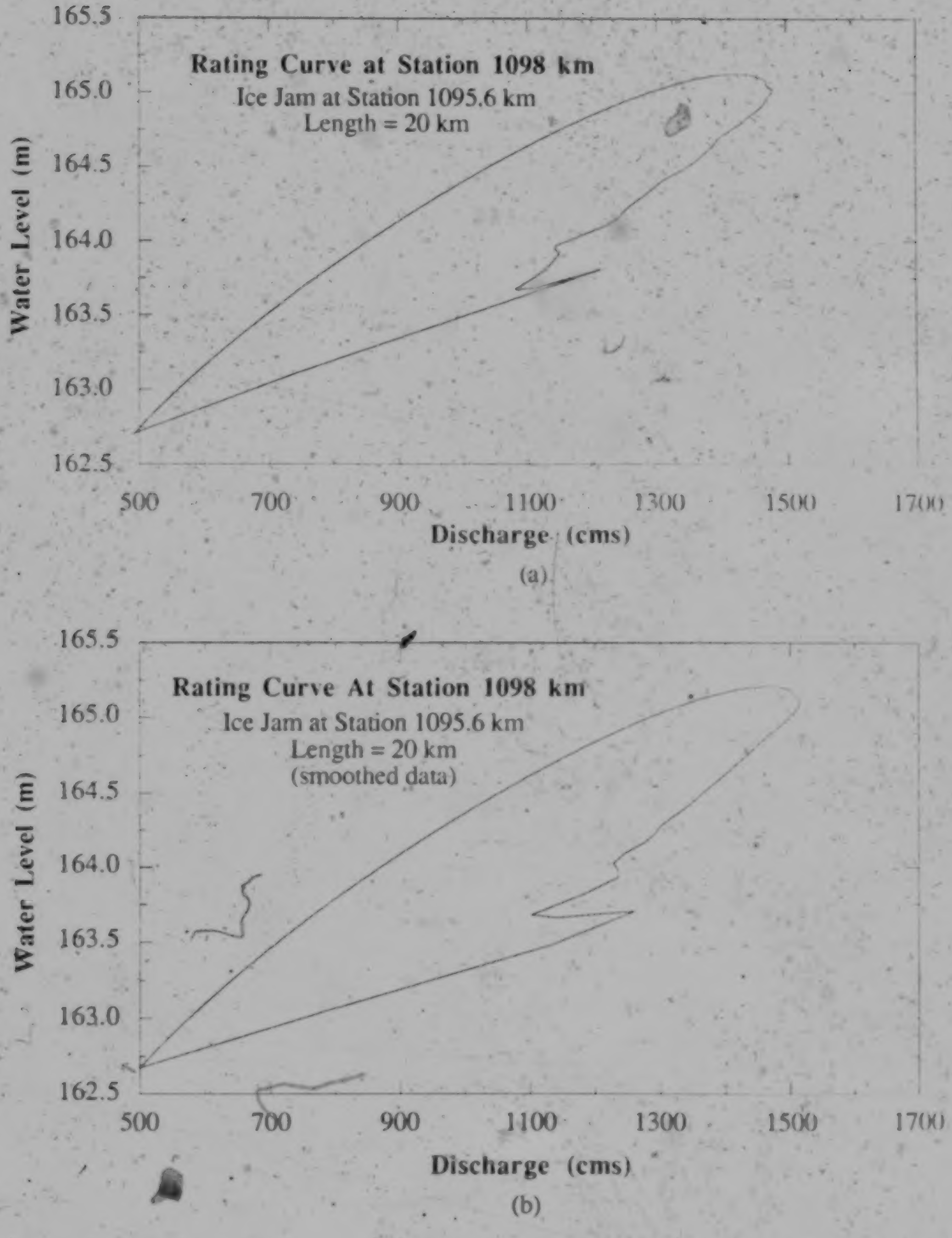
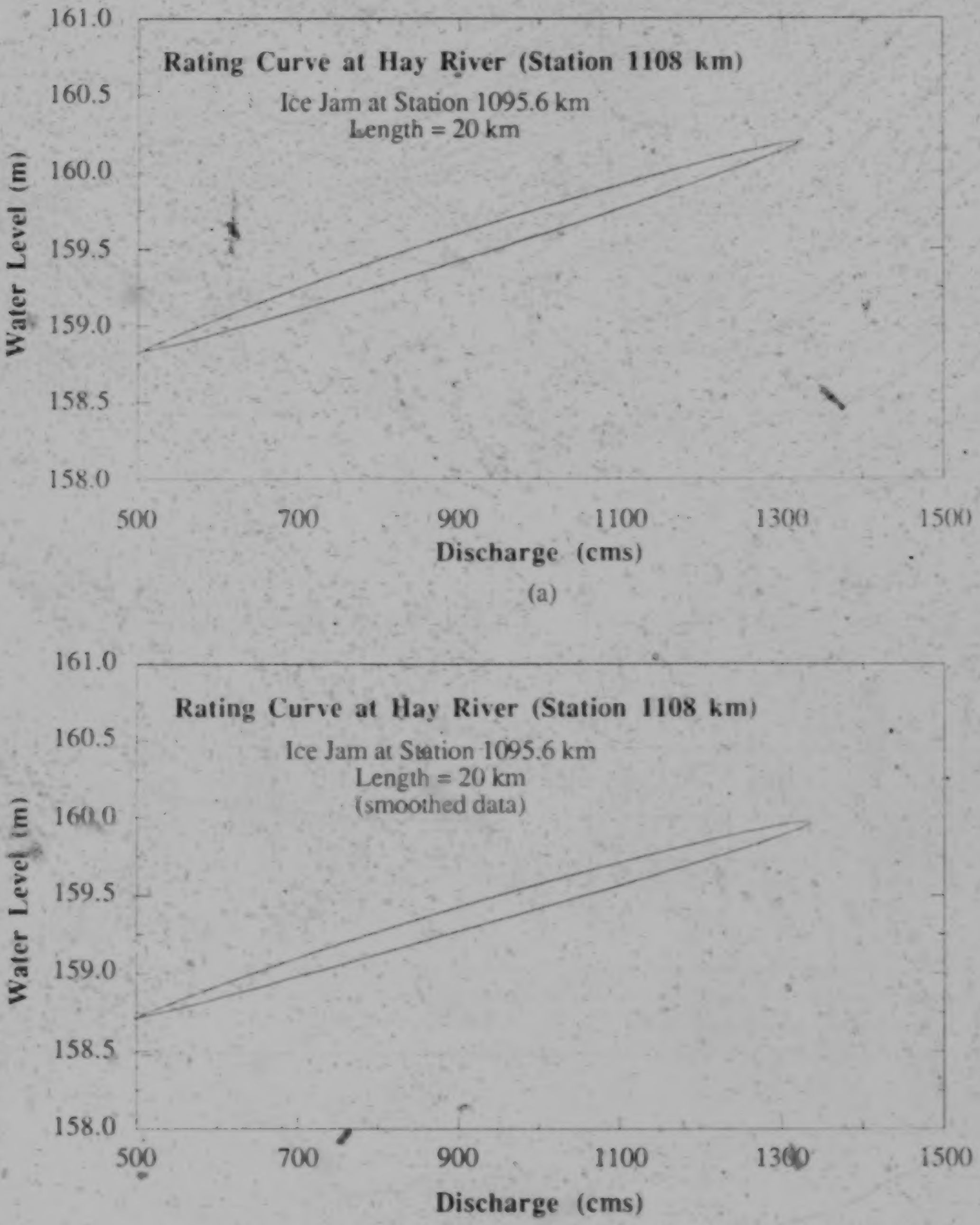


Figure 6.22 Rating curves at Station 1098 for the jam released at Station 1095.6 km, (a) NTS map data, (b) smoothed data



(a)

(b)

Figure 6.23 Rating curves at Hay River for the jam released at Station 1095.6 km, (a) NTS map data, (b) smoothed data.

7. CONCLUSIONS

This study, provides a comprehensive examination of the state of the art in Finite Element modeling of unsteady open channel flow. Although both the Finite Difference Method and the FEM have received considerable attention in this area of research, from a practical point of view, no one numerical scheme has emerged as generally applicable to all flow situations. Therefore, the objective of this study was to determine if a numerical technique could be developed which would provide accurate and stable solutions to the one-dimensional open channel flow equations for a wide range of flow situations.

The finite element method was used for this study because of its underlying consistency and generality, which were particularly suited to the implementation and comparison of a variety of numerical techniques. Those finite element schemes investigated were the Taylor-Galerkin method; the Dissipative-Galerkin method; the Characteristic-Dissipative-Galerkin method; and the Least-Squares method. Variations of some of these methods were considered as well: This included a lumped Taylor-Galerkin and a symmetric formulation for the Characteristic-Dissipative-Galerkin scheme. For illustrative purposes results were compared not only with the Bubnov-Galerkin method, but also the common four-point implicit box finite difference scheme.

Initially, a linear stability analysis was used to examine the amplification and phase characteristics of the various numerical methods on the linearized St. Venant equations. Although this linear analysis did not predict instabilities associated with nonlinearities in the problem or the effects of gradient boundary conditions, it did provide a valuable tool not only as a basis for a comparison of the various numerical schemes but also to determine the appropriate discretization of a problem for a particular method. The numerical schemes were then tested extensively for linear and non-linear problems, as well as for steady and unsteady flow situations. The steady flow tests included the case of an hydraulic jump in a horizontal, frictionless flow and the computation of gradually varied flow profiles, for combinations of mild and steep channels. To examine unsteady flow situations a linear formulation was used first. The test consisted of a single surface perturbation propagated in a horizontal frictionless channel (Froude number = 0). This disturbance was modeled as it split into two separate waves, each propagated at the characteristic velocities $\left(U \pm c\right)$. This test was then extended to examine the non-linear equations and a variety of Froude numbers (0.0, 0.5, 1.0, and 2.0). Further unsteady flow tests included a classic dambreak problem as well as the propagation of a wave through an hydraulic jump. Finally, the various models were tested for kinematic and diffusive wave problems.

In the limiting case where the dynamic terms dominate, the linear stability analysis indicated that the TG (explicit) method would perform well at low Courant numbers though, as reported by other investigators (Selmin *et al*, 1985), the stability interval was reduced below the unit Courant-Fredrichs-Lewy condition. When tested on the hydraulic jump problem, both the TG and the lumped TG schemes exhibited spurious oscillations in the vicinity of the shock. In propagating disturbances both were relatively stable, with the lumped version displaying slightly poorer phase accuracy. Although this explicit scheme has the distinct advantage of not requiring as much computational effort as an implicit method, its relatively poor performance in the vicinity of shocks and its reduced stability range, limits its general applicability. This time step restriction was of particular importance to steady flow problems, where it was desirable to step quickly to the ultimate steady-state solution. Thus, the implicit schemes appeared more versatile in terms of handling both steady and unsteady flows with a single model.

For the fully dynamic case, the linear stability analysis indicated that the LS method did not sufficiently dampen the very short wavelengths. As a result, it was subject to oscillations in the vicinity of the hydraulic jump, especially as the Courant number increased. In addition, this scheme was not particularly selective in damping the short wavelengths and consequently, when propagating disturbances, it was found to dissipate the peak more than the TG scheme. In fact, a semi-implicit formulation of the LS scheme was found to be even more dissipative than the BFD method run at an implicitness of $\theta = 0.6$. Finally this method proved unstable when critical or supercritical flows were concerned.

The linear stability analysis showed that the DG and CDG schemes displayed the highly desirable combination of selective damping of high frequency progressive waves over a wide range of Courant numbers, and good phase accuracy at low Courant numbers, when a semi-implicit formulation was used. The DG method appeared to be marginally more dissipative than the CDG scheme for these progressive waves. However the CDG scheme, which considers both waves in quantifying the upwind weighting, was able to achieve a higher degree of damping of high frequency regressive waves than the DG method.

Both the DG and CDG methods proved stable for the hydraulic jump problem, although the CDG was noticeably freer of spurious oscillations. It was found, however, that oscillations in the DG solution could be mitigated by increasing the amount of upwind weighting. These two methods were also found to perform well in the other tests, propagating disturbances without significant damping of the peak and displaying good phase accuracy. However, when considering the propagation of a wave through an hydraulic jump, the DG method was again subject to spurious oscillations in the vicinity of flow discontinuities, unless the upwind weighting was increased.

It is important to note that the upwind weighting for both of these schemes depends entirely upon the dynamic terms in the equations. Therefore, because the practical application of these models would also necessitate their use on diffusive or kinematic wave problems, the stability analysis was extended to consider the limiting case of kinematic waves, as well. For a fully kinematic wave both the DG and CDG methods displayed less damping of wavelengths greater than 5Δx than the BFD method, though their phase accuracy was comparable. Both methods also exhibited selective damping of most short wavelengths. However, they were completely non-dissipative for the 2Δx waves. Thus, it was found that an additional constraint on the spatial discretiztion would be required to avoid an unstable solution when modeling a kinematic wave with these schemes. Fortunately, this constraint was not found to be overly-restrictive in terms of practical situations.

Because the application of the CDG method to symmetric systems of hyperbolic equations assures orthogonality of the eigenvectors of the upwinding matrix, the investigation of this scheme was extended to examine a symmetric form of the St. Venant equations. This allowed for the determination of whether or not the cross-upwinding effect was significant enough to require the use of symmetric forms in general shallow flow modeling. As this necessitated the use of a non-conservation system, a comparison of conservative and non-conservative non-symmetric formulations was also done. Although cross-upwinding effects appeared insignificant, it was found that the results obtained using the non-conservation system did produce quite different results in the non-symmetric form. Thus it was concluded that the cross-upwinding effects, if they were substantial, could not be remedied thorough the use of a symmetric formulation, as the non-conservation form appeared to counteract any beneficial effects afforded by the symmetric approach. As a result the non-symmetric conservation form of the St. Venant equation were deemed quite suitable for one-dimensional flow problems. In considering extensions to two-dimensional flow, where a non-conservation form may be desirable, it might be advisable to consider the symmetric formulation to counteract the detrimental effects associated with the non-conservation form of the equations.

The CDG scheme was then applied to the determination of steady gradually varied flow profiles as well as the propagation of ice jam release surges over a 500 kilometer reach of the Hay River in Alberta/Northwest Territories. This combined the complications of friction, varying channel width, and subcritical and supercritical flows all into one test. In the unsteady case, this created a dambreak type of problem which began as a very dynamic situation and then developed into an essentially kinematic wave problem as this disturbance propagated downstream. Again, the CDG scheme performed well for both transient and steady flows, even in the vicinity of flow discontinuities, without requiring additional upwind weighting.

This study, provides a comprehensive examination of the state of the art in finite element modeling of unsteady open channel flow. By employing an objective and critical analysis through the use of exhaustive and rigorous tests, considerable evidence has been presented illustrating the applicability of finite element methods to open channel flow problems, including essentially kinematic wave problems. Furthermore, a reconsideration of the fundamental role of the characteristics in the determination of the upwind weighting have lead to the refinement of the DG method, producing the CDG scheme, a method which has been shown to perform well over a wide range of steady and unsteady open channel flow problems. This refinement requires no additional computational expense and is relatively simple to implement. Moreover, its fundamental consideration of the characteristics in quantifying the upwind weighting indicates that the CDG method may be extended with confidence to other hyperbolic systems such as: two-dimensional flows; multi-layer fluids; or sediment transport problems.

REFERENCES

Ames, W. F., 1977, Techniques in Partial Differential Equations (2 ed.). Academic Press, Toronto, 365 pages.

Abbott, M. B., 1979, Computational Hydraulics - Elements of the Theory of Free Surface Flows. Pitman Publishing Company, London, England, 324 pages.

Amien, M. , 1968, "An Implicit Method for Natural Flood Routing". Water Resources Research, Vol. 4, No. 4, pp. 719-726.

Amien, M. and H. L. Chu, 1974, "Implicit Numerical Modeling of Unsteady Flows". ASCE Journal of Hydraulic Engineering, Vol. 101 (HY6), pp. 717-731.

Baker, A. J. and J. W. Kim, 1987, "A Taylor Weak-Statement Algorithm for Hyperbolic Conservation Laws". International Journal for Numerical Methods in Fluids, Vol. 7, pp. 489-520.

Brooks, A. N. and T. J. R. Hughes, 1982, "Streamline Upwind/Petrov-Galerkin Formulations for Convection Dominated Flows with Particular Emphasis on the Incompressible Navier-Stokes Equations". Computer Methods in Applied Mechanics and Engineering, Vol. 32, pp. 199-259.

Burnett, D. S. , 1987, Finite Element Analysis. Addison Wesley Publishing Company, Don Mills, Canada, 440 pages.

Butkov, E. , 1968, Mathematical Physics. Addison-Wesley Publishing Company, Reading, Massachusetts, 735 pages.

Carey, G. F. and B. N. Jiang, 1988, "Least-Squares Finite Element for First-Order Hyperbolic Systems". International Journal for Numerical Methods in Engineering, Vol. 26, pp. 81-93.

Chester, C. R. , 1971, Techniques in Partial Differential Equations. McGraw-Hill Book Company, Toronto, Canada, 440 pages.

Cunge, J.A.,, F. M. Holly, Jr., and A. Verwey, 1980, Practical Aspects of Computational River Hydraulics. Pitman Publishing Company, London, England, 420 pages.

Chow, V. T., 1959, Open-Channel Hydraulics. McGraw-Hill Book Company, New York, N. Y., 680 pp.

Donea, J., 1984, "A Taylor-Galerkin Method for Convective Transport Problems". International Journal for Numerical Methods in Engineering, Vol. 20, pp. 101-119.

Dull R. W. and R. Dull, 1951, Mathematics for Engineers. McGraw-Hill Book Company, New York, N. Y., 822 pp.

Fennema, R. J., and M. H. Chaudhry, 1987, " Simulation of One-Dimensional Dam-Break Flows". International Journal of Hydraulic Research, Vol. 25, No. 1, pp. 41-51.

Fread, D. L., 1988, The NWS DAMBRK Model: Theoretical Background/ User Documentation. Office of Hydrology, National Weather Service (NMS), Maryland, USA.

Froehlich, D. C., 1985, a discussion of "A Dissipative Galerkin Scheme for Open-Channel Flow". N. D. Katapodes ASCE Journal of Hydraulic Engineering, Vol. 111 (HY4), pp. 1200-1204.

Gerard, R. and S. Stanley, 1988, Hay River Study Phase I - Ice Jams and Flood Forecasting. Water Resources Engineering Report No. 88-6, Department of Civil Engineering.

Gerard, R., Hicks, F.E. and M. Jasek, 1990, Hay River Study Phase II - Ice Jams and Flood Forecasting. Water Resources Engineering Report No. 90-4, Department of Civil Engineering.

Gresho, P. M. and R. L. Lee, 1987, "Comments on the Group Velocity of Some Numerical Schemes". International Journal for Numerical Methods in Fluids, Vol. 7, pp. 1357-1362.

Hildebrand, F. B., 1966, Advanced Calculus for Applications. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 733 pp.

Henderson, F. M., 1966, Open Channel Flow. MacMillan Publishing Co., Inc., New York, N. Y., 522 pp.

Hood, P. and C. Taylor, 1974, "Navier-Stokes Equations Using Mixed Interpolation". Finite Element Methods in Flow Problems: International Symposium, Swansea, [illegible]. K. Proceedings: J. T. Oden, O. C. Zienkiewicz, R. H. Gallagher and C. Taylor, Eds. University of Alabama Huntsville Press, Huntsville, AL, pp. 121-132.

Hughes, T. J. R., 1978, "A Simple Scheme for Developing 'Upwind' Finite Elements". International Journal for Numerical Methods in Engineering, Vol. 12, pp. 1359-1365.

Hughes, T. J. R.and A. Brooks, 1980, "Galerkin/Upwind Finite Element Mesh Partitions in Fluid Mechanics". Boundary and Interior Layers-Computational and Asymptotic Methods, J. J. H. Miller, Ed., Vol. 1, Boole, Dublin, pp. 103-112.

Hughes, T. J. R. and T. E. Tezduyar, 1984, "Finite Element Method for First-Order Hyperbolic Systems with Particular Emphasis on the Compressible Euler Equations". Computer Methods in Applied Mechanics and Engineering, Vol. 45, pp. 217-284.

Hughes, T. J. R., L. P. Franca and M. Mallet, 1986, "A New Finite Element Formulation for Computational Fluid Dynamics: I. Symmetric Forms of the Compressible Euler and Navier-Stokes Equations and the Second Law of Thermodynamics". Computer Methods in Applied Mechanics and Engineering; Vol. 54, pp. 223-234.

Hughes, T. J. R., M. Mallet and A. Mizukami, 1986, "A New Finite Element Formulation for Computational Fluid Dynamics: II. Beyond SUPG". Computer Methods in Applied Mechanics and Engineering, Vol. 54, pp. 341-355.

Hughes, T. J. R. and M. Mallet, 1986, "A New Finite Element Formulation for Computational Fluid Dynamics: III. The Generalized Streamline Operator for Multidimensional Advective-Diffusive Systems". Computer Methods in Applied Mechanics and Engineering, Vol. 58, pp. 305-328.

Jiang, B. N. and L. A. Povinelli, 1989, "Least-Squares Finite Element Method for Fluid Dynamics". Proc. 7th International Conference on Finite Elements in Fluids, University of Alabama, Huntsville, pp. 105-110.

Jiang, B. N. and G. F. Carey, 1988, "A Stable Least-Squares Finite Element Method for Non-Linear Hyperbolic Systems". International Journal for Numerical Methods in Fluids, Vol. 8, pp. 933-942.

Katopodes, N. D., and T. Strelkoff, 1979, "Two Dimensional Shallow Water Wave Models". ASCE Journal of Hydraulic Engineering, Vol. 105 (HY2), pp. 317-334.

Katopodes, N. D., 1984, "A Dissipative Galerkin Scheme for Open-Channel Flow". ASCE Journal of Hydraulic Engineering, Vol. 110 (HY4), pp. 450-466.

Katopodes, N. D., 1984, "Fourier Analysis of Dissipative FEM Channel Flow Model". ASCE Journal of Hydraulic Engineering, Vol. 110 (HY7), pp. 927-944.

Katopodes, N. D. and C. T. Wu, 1986, "Explicit Computation of Discontinuous Channel Flow". ASCE Journal of Hydraulic Engineering, Vol. 112 (HY6), pp. 456-475.

Katopodes, N. D., 1987, "Analysis of Transient Flow Through Broken Levees". Turbulence Measurements and Flow Modeling, C. J. Chen, L. D. Chen and F. M. Holly Jr., Eds. pp. 301-310.

Kellerhals, R., C. R. Neill and D. I. Bray, 1962, Hydraulic and Geomorphic Characteristics of Rivers in Alberta. River Engineering and Surface Water Hydrology Report 72-1, Research Council of Alberta, Edmonton, 54 pp.

Keulegan, G. B., 1938, "Laws of Turbulence in Open Channels". Journal of Research of the National Bureau of Standards, U. S. Dept. of Commerce, Vol. 21, pp. 707-741.

Kinnmark, I. P. E. and W. G. Gray, 1982. "Time-Weighting of the Momentum Equation in Explicit Wave Equation Models of Surface Water Flow". Proc. 4th Intl. Conf. on Finite Elements in Water Resources, K. P. Holz, U. Meissner, W. Zielke, C. A. Brebbia, G. Pinder, and W. Gray, Eds., Berlin, pp. 5.67-5.77.

Kinnmark, I. P. E., 1984, "The Shallow Water Wave Equations: Formulation, Analysis and Application". Lecture Notes in Engineering, C. A. Brebbia and S. A. Orszag, Eds., Berlin, 187 pp.

Lapidus, L. and Pinder G.F., 1982, Numerical Solution of Partial Differential Equations in Science and Engineering. John Wiley and Sons, Toronto, Canada, 677 pages.

Lee, J. K., 1987, a discussion of "Fourier Analysis of Dissipative FEM Channel Flow Model". N. D. Katapodes ASCE Journal of Hydraulic Engineering, Vol. 113 (HY2), pp. 267-270.

Lee, J. K. and D. C. Froehlich, 1986, Review of the Literature on the Finite-Element Solution of the Equations of Two-Dimensional Surface-Water Flow in the Horizontal Plane. U. S. Geological Survey Circular 1009, pages.

Lee, J. H. W., J. Peraire and O. C. Zienkiewicz, 1987, "The Characteristic-Galerkin Method for Advection-Dominated Problems - An Assessment". Computer Methods in Applied Mechanics and Engineering, Vol. 61, pp. 359-369.

Liggett, J. A. and J. A. Cunge, 1975, "Numerical Methods of Solution of the Unsteady Flow Equations in Open Channels". Unsteady Flow in Open Channels, K. Mahmood and Y. Yevjevich, Eds., Vol. 1, Water Resources Publications, Fort Collins, Colorado, pp. 89-182.

Linsley, R. K., M. A. Kohler and J. L. H. Paulhus, 1982, Hydrology for Engineers (3 ed.). McGraw-Hill Book Company, New York, N. Y., 482 pages.

Lyn, D. A. and P. Goodwin, 1987, "Stability of a General Priessman Scheme". ASCE Journal of Hydraulic Engineering, Vol. 113 (HY1), pp. 16-28.

Lynch, D. R. and W. G. Gray, 1979, "A Wave Equation Model for Finite Element Tidal Computations". Computers and Fluids, Vol. 7, No. 3, pp. 207-228.

Miller, A and J. A. Cunge., 1975, "Simplified Equations of Unsteady Flow". Unsteady Flow in Open Channels, K. Mahmood and V. Yevjevich, Eds., Vol. 1, Water Resources Publications, Fort Collins, Colorado, pp. 183-258.

Ponce, V. M. and D. B. Simons, 1977, "Shallow Water Wave Propagation in Open Channel Flow". ASCE Journal of Hydraulic Engineering, Vol. 103 (HY12), pp. 1461-1476.

Ponce, V. M. , R. M. Li and D. B. Simons, 1978, "Applicability of Kinematic and Diffusion Wave Models". ASCE Journal of Hydraulic Engineering, Vol. 104 (HY3), pp. 947-958.

Ponce, V. M. , H. Indlekofer and D. B. Simons, 1978, "Convergence of Four-Point Implicit Water Wave Models". ASCE Journal of Hydraulic Engineering, Vol. 104 (HY7), pp. 947-958.

Ponce, V. M. , H. Indlekofer and D. B. Simons, 1979, "The Convergence of Implicit Bed Transient Models". ASCE Journal of Hydraulic Engineering, Vol. 105 (HY4), pp. 351-362.

Preissmann, A., 1961, "Propagation des inntumescences dans les cannaux et Les Rivieres". 1e Congress de L'Association Francaise de Calcule, Grenoble,France, pp. 433-442.

Raymond, W. H. and A. Garder, 1976, "Selective Damping in a Galerkin Method for Solving Wave Problems with Variable Grids". Monthly Weather Review, Vol. 104, pp. 1583-1590.

Richtmyer, R. D. and K. W. Morton, 1967, Difference Methods for Initial Value Problems, (2 ed.). Interscience Publishers, New York, N. Y., 405 pages.

Salas, S. L. and E. Hille, 1982, Calculus One and Several Variables with Analytic Geometry, (4 ed.). Tom Wiley and Sons, New York, N. Y., ??? pp.

Samuels, P. G., 1985, Modelling of River and Flood Plain Flow Using the Finite Element Method, Research Report No. SR 61, Hydraulics Research Ltd., Wallingford, England, 198 pages.

Schlichting, H., 1979, Bounday Layer Theory, (7 ed.). McGraw-Hill Book Company, New York, N. Y., 817 pp.

Selmin, V., J. Donea and L. Quartapelle, 1985, "Finite Element Methods for Non-Linear Advection". Computer Methods in Applied Mechanics and Engineering, Vol. 52, pp. 817-845.

Stasa, F. L., 1985, Applied Finite Element Analysis for Engineers. Holt, Rinehart and Winston, New York, N.Y., 657 pages.

Streeter, V. L.and E. B. Wylie, 1981, Fluid Mechanics. McGraw-Hill Ryerson Ltd., Toronto, 562 pp.

Vichnevetsky, R., 1987, "Wave Propagation analysis of Difference Schemes for Hyperbolic Equations: A Review". International Journal for Numerical Methods in Fluids, Vol. 7, pp. 409-452.

Walters, R. A. and G. F. Carey, 1983, "Analysis of Spurious Oscillation Modes for the Navier-Stokes and Shallow Water Equations". Computer and Fluids, Vol. 11, No. 1, pp. 51-68.

Wang, S. S. and S. E. Adeff, 1987, "A Depth Integrated Model for Solving Navier-Stokes Equations Using Dissipative Galerkin Scheme". Turbulence Measurements and Flow Modeling, C. J. Chen, L. D. Chen and F. M. Holly Jr., Eds. pp. 311-321.

White, F. M., 1986, Fluid Mechanics, (2 ed.). McGraw-Hill Book Company, New York, N. Y., 732 pp.

Yevjevich V., 1975, "Introduction". Unsteady Flow in Open Channels, K. Mahmood and V. Yevjevich, Eds., Vol. 1, Water Resources Publications, Fort Collins, Colorado, pp. 1-27.

Zienkiewicz, O. C.,1977, The Finite Element Method. McGraw-Hill Book Company (UK) Ltd., Maidenhead, Berkshire, England, 787 pages.

APPENDIX A

BFD Program Listing

Files included are **Page**

fe.h....364
femain.c....366
fecore.c....368
fecom.c....376
fegauss.c....380
festuff.c....381
feio.c....383
fetest.c....383
feice.c....383
fewindow.c....383

fe.h

```
#include "math.h"
#include "stdio.h"
#include "stdlib.h"

#define  MNSF      16
#define  MRNSF      4
#define  MDOF      36
#define  MGPTS      9
#define  NDIMS      2
#define  NPARAMS    3
#define  NBPARAMS   7
#define  NVARS      2
#define  NKEQNS    20

struct shapefuncs {
        int                       dof ;
        double                    f[MNSF] ;
        double                    dfdr[MNSF] ;
        double                    dfds[MNSF] ;
        double                    dfdt[MNSF] ;
        double                    d2fdr[MNSF] ;
        } ;

struct govfuncs {
        int                       dof ;
        double                    detJ ;
        double                    w ;
        double                    p[NPARAMS] ;
        double                    f[MNSF] ;
        double                    dfdx[MNSF] ;
        double                    dfdy[MNSF] ;
        double                    dfdz[MNSF] ;
        double                    d2fdx[MNSF] ;
        } ;

struct gausspts {
        double                    x ;
        double                    y ;
        double                    z ;
        double                    w ;
        } ;

struct  node  {
        int                       n ;
        int                       i ;
        int                       ui ;
        int                       fxc          ;
        double                    x[NDIMS]     ;
        double                    u[NVARS]     ;
        double                    p[NPARAMS]   ;
        double                    uo[NVARS]    ;
        double                    uoe[NVARS]   ;
        float                     f[NDIMS]     ;
        struct node               *nextnp      ;
        } ;

struct  element  {
        int                       n            ;
        int                       vtype        ;
        int                       gtype        ;
        int                       nnds         ;
        struct node               *nps[MNSF];
        double                    p[NPARAMS] ;
        double                    *matrices ;
        float                     elinfo ;
        struct element            *nextelp ;
        } ;

struct  belement  {
        int                       n            ;
        int                       vtype        ;
        int                       gtype        ;
        int                       nnds         ;
        struct node               *nps[MRNSF];
        double                    p[NBPARAMS];
        double                    *matrices ;
        int                     bcs[NVARS];
        struct belement         *nextbelp ;
        } ;

struct control {
        int                     trans      ;
        int                     dims       ;
        int                     vars       ;
        int                     params     ;
        int                     bparams ;
        int                     Keqns[NKEQNS];
        int                     nodes      ;
        int                     elms       ;
        int                     belms      ;
        int                     frnds      ;
        int                     ukns       ;
        int                     sym        ;
        int                     nonlin     ;
        } ;

struct eqnset {
        double                  *Kp ;
        double                  *Lp ;
        double                  *Fp ;
        long                    *diagp ;
        int                     neqns ;
        } ;

struct pointers {
        struct node             *N ;
        struct element          *El ;
        struct belement         *B ;
        struct node             **iptrs ;
        double                  *S ;
        double                  *M ;
        }  ;

struct elmpointers {
        double                  *K ;
        double                  *F ;
        double                  *S ;
        double                  *M ;
        double                  *J ;
        int                     n  ;
        }  ;

struct fxnodes {
        int                     i ;
        int                     j[MDOF] ;
        double                  k[MDOF] ;
        double                  f ;
        } ;

struct transient {
        int                     nsteps ;
        int                     method ;
        int                     iter ;
        double                  dtfac ;
        double                  t ;
        double                  dt ;
        double                  theta ;
        } ;

struct   RegMesh {
        int                     nx ;
        int                     ny ;
        int                     nbx ;
        int                     nby ;
        int                     elsinbx[MNSF] ;
        int                     elsinby[MNSF] ;
        int                     eltype ;
        int                     maptype ;
        struct control          con ;
        struct blockmap         *firstmap ;
        } ;

struct blockmap {
        float                   xnodes[MNSF] ;
        float                   ynodes[MNSF] ;
        } ;
```

femain.c

```
#include        "fe.h"
main(argc,argv)
int  argc ;
char **argv ;
{
    char      ain ;
    int  nin, errcode, resfilenum, i ;

    initXwindow(argc,argv) ;
    init_gp() ;
    printf(" Input command (q,i,R,Q,o,a,s,d,I,S,T,p,u) and number\n");
    ain = 't' ;
    while(ain != 'q'){
        printf("> ");
        scanf(" %c %d",&ain,&nin) ;
        switch (ain) {

            case 'i' :
                errcode = input(nin) ;
                ScaleXwindow(0);
                break ;
            case 'R' :
                ScaleXwindow(nin);
                break;
            case 'Q' :
                SetAuxPlot(nin);
                ScaleXwindow(2);
                break;
            case 'o' :
                errcode = output(nin) ;
                break ;
            case 'a' :
                errcode = assemble(nin,0,0,2,0) ;
                printf(" assemble code = %d\n",errcode);
                break ;
            case 's' :
                errcode = solve(nin,1,0) ;
                break ;
            case 'd' :
                errcode = test_trans(nin) ;
                break ;
            case 'I' :
                errcode = test_ice(nin) ;
                ScaleXwindow(2) ;
                break ;
            case 'S' :
                errcode = test_smooth(nin) ;
                ScaleXwindow(2) ;
                break ;
            case 'T' :
                errcode = test_time(nin) ;
                break ;
            case 'p' :
                errcode = set_param(nin) ;
                break ;
            case 'u' :
                errcode = set_ptou(nin) ;
                break ;
        }
    }
    killXwindow();
    exit(0);
}
FILE      *get_fptr(code)
int       code ;
{
    FILE      *fp, *fopen() ;
    char      s[63] ;

    if(code == 1) {
        printf("\n Input Data File Name  ") ;
        scanf(" %s",s) ;
        fp = fopen(s,"r") ;
    }
    else
        fp = stdin ;
    return( fp ) ;
}
FILE      *put_fptr(code)
int       code ;
{
    FILE      *fp, *fopen() ;
    char      s[63] ;

    if(code == 1) {
        printf("\n Input Output File Name  ") ;
        scanf(" %s",s) ;
        fp = fopen(s,"w") ;
    }
    else {
        if(code == 2)
            fp = fopen("fe.out","w") ;
        else
            fp = stdin ;
    }
    return( fp ) ;
}
int       UpDateMes(astr,bstr, cstr, dstr)
char          *astr, *bstr, *cstr, *dstr ;
{
    printf("%s\n",dstr) ;
    return(0);
}
int       EndMes()
{
    return(0);
}
```

fecore.c

```
#include "fe.h"
#define  KE(A,B)  ( *(ep.K + ep.n*(A) + (B)) )
#define  SE(A,B)  ( *(ep.S + ep.n*(A) + (B)) )
#define  ME(A,B)  ( *(ep.M + ep.n*(A) + (B)) )
#define  JE(A,B)  ( *(ep.J + ep.n*(A) + (B)) )
#define  FE(B)  ( *(ep.F + B) )
#define  GRAV      9.81
struct control     N ;
struct pointers    gp;
struct RegMesh     Mesh ;
struct fxnodes     *first_fnp, *next_fnp ;
struct transient   tvals ;
int                Nukns, Nrfixed,  Fac, *BFnodes, BandWidth ;
struct elmpointers  ep ;
struct eqnset      eqnsets[4] ;
double             uchange, *fprime ;
extern double      defndp , defelp ;
int                init_gp()
{
    int i ;

    gp.N = NULL ;
    gp.El = NULL ;
    gp.B = NULL ;
    gp.S = NULL ;
    gp.M = NULL ;
    for(i=0;i<4;i++) {
        eqnsets[i].Kp = NULL ;
        eqnsets[i].Fp = NULL ;
        eqnsets[i].diagp = NULL ;
        eqnsets[i].neqns = 0 ;
    }
    ep.K = NULL ;
    ep.F = NULL ;
    ep.S = NULL ;
    ep.M = NULL ;
    ep.J = NULL ;
    fprime = NULL ;
    first_fnp = NULL ;
    Mesh.nx = 10 ;
    Mesh.ny = 1 ;
    Mesh.nbx = 1 ;
    Mesh.nby = 1 ;
    Mesh.eltype = 111 ;
    Mesh.maptype = 111 ;
    Mesh.firstmap = (struct blockmap *) calloc(24,sizeof(struct blockmap)) ;
    tvals.nsteps = 1 ;
    tvals.t = 0.0 ;
    tvals.dt = 1.0 ;
    tvals.theta = 0.5 ;
    tvals.dtfac = 1.0 ;
    N.sym = 1 ;
    N.nonlin = 1 ;
    BandWidth = 4 ;
}

int   * update_PGKe(gbfp,gtfp,nbf,ntf,np,vcode,theElp )
int                            nbf, ntf, vcode  ;
struct element          *theElp ;
struct node             *np[] ;
struct govfuncs         *gbfp, *gtfp ;
{
    int i, j, err, n ;
    double  Qnew, Anew, hnew, Unew, So, ff, w, cel, dx, tdt, mtdt,R,Cstar,dw ;

    if(vcode<0)
        n = N.vars ;
    else
        n = 1 ;

/*  Compute Parameters Here */

    Qnew  = 0.0 ;
    Anew  = 0.0 ;
    Unew  = 0.0 ;
    So = 0.0 ;
```

```
     ff = 0.0 ;
     w = 0.0 ;
     dx = np[1]->x[0] - np[0]->x[0] ;
     dw = np[1]->p[2] - np[0]->p[2] ;
     tdt = tvals.theta * tvals.dt ;
     mtdt = tvals.dt * (1.0 - tvals.theta) ;

     for(i=0;i<nbf;i++) {
          Anew += gbfp->f[i]*np[i]->u[0] ;
               Qnew += gbfp->f[i]*np[i]->u[1] ;
          w += gbfp->f[i]*np[i]->p[2] ;
          ff += gbfp->f[i]*np[i]->p[1] ;
     }
     Unew = Qnew/Anew ;
     hnew = Anew/w ;
     R = Anew/(w+hnew+hnew) ;
     if (ff>0){
          Cstar = 5.75 * log10(fabs(R/ff)) + 6.2 ;
          ff = fabs(Unew)/R/Cstar/Cstar ;
     }
     So = -(np[1]->p[0] - np[0]->p[0])/dx ;
     So = So +(dw*hnew/w /dx) ;

/*   for linear wave eq'n only
     Unew = 2.93 ;
     Unew = 0.0 ;
     hnew = 3.5 ;
     ff = 0.0 ;
*/

     cel = sqrt(fabs(GRAV*hnew)) ;

/*   subcritical flow */
          SE(1,0) = 1. ;
          SE(1,1) = -tdt * 2./dx ;
          SE(1,2) = 1. ;
          SE(1,3) = tdt *  2./dx ;
          SE(2,0) = tdt*(-(GRAV * So) - (2 * (cel*cel - Unew*Unew)/dx)) ;
          SE(2,1) = 1. + tdt * (-(4. * Unew/dx) + ff) ;
          SE(2,2) = tdt*(-(GRAV * So) + (2 * (cel*cel - Unew*Unew)/dx)) ;
          SE(2,3) = 1. + tdt * ( (4. * Unew/dx) + ff) ;

          FE(0) = 0. ;
          FE(1) =  np[0]->uo[0] + np[1]->uo[0] + 2*mtdt*np[0]->uo[1]/dx - 2*mtdt*np[1]->uo[1]/dx;
          FE(2) =  mtdt * ( ( 2*(cel*cel - Unew*Unew)/dx) + (GRAV * So) ) * np[0]->uo[0] + mtdt * ( (-
               2*(cel*cel - Unew*Unew)/dx) + (GRAV * So) ) * np[1]->uo[0]
               + ( 1. - mtdt * (-(4. * Unew/dx) + ff) ) * np[0]->uo[1]
               + ( 1. - mtdt * ( (4. * Unew/dx) + ff) ) * np[1]->uo[1];
          FE(3) = 0. ;


/* Incorporate boundary conditions directly here */
    if(theElp->n == 1) {
    /*    A given at upstream end
          SE(0,0) = 1. ;
          FE(0) = np[0]->uo[0] ;
    */
    /*   Q given at upstream end      */
          SE(0,0) = SE(1,0) ;
          SE(0,1) = SE(1,1) ;
          SE(0,2) = SE(1,2) ;
          SE(0,3) = SE(1,3) ;
          FE(0) = FE(1) ;
          SE(1,0) = 0. ;
          SE(1,1) = 1. ;
          SE(1,2) = 0. ;
          SE(1,3) = 0. ;
          FE(1) = np[0]->uo[1] ;
    }

    if(theElp->n == N_elms) {
    /*   Q given at downstream end
          SE(3,3) = 1. ;
          FE(3) = np[1]->uo[1] ;
    */
    /*   A given at downstream end      */
          SE(3,0) = SE(2,0) ;
          SE(3,1) = SE(2,1) ;
          SE(3,2) = SE(2,2) ;
          SE(3,3) = SE(2,3) ;
          FE(3) = FE(2) ;
```

```
        SE(2,0) = 0. ;
        SE(2,1) = 0. ;
        SE(2,2) = 100000000. ;
        SE(2,3) = 0. ;
        FE(2) = np[1]->uo[0]*100000000. ;

    }

            if(N.nonlin > 0) {
                JE(0,0) =  0;
                JE(0,1) =  0;
                JE(0,2) = 0 ;
                JE(0,3) =  0;
                JE(1,0) =  0;
                JE(1,1) = 0 ;
                JE(1,2) = 0 ;
                JE(1,3) = 0 ;

            }

return(0) ;
}

int     transient(nvar,elcode,tol)
int     nvar,elcode ;
double  tol ;
{
    int     i, j, jj, itnum, day, hr, min, sec, hnum ;
    long    time ;
    struct  node    *nodep, *maxnodep ;
    FILE    *fptr, *put_fptr() ;


    N.trans = 1 ;

    set_ptou(0) ;
    if((fptr = put_fptr(1) ) != NULL){
        fprintf(fptr,"Non-conservation form (y and q)\n") ;
        fprintf(fptr,"Time Step =%g\tTheta = %g\n",
            tvals.dt,tvals.theta) ;
        fprintf(fptr,"\n") ;

        for(i=0;i<tvals.nsteps;i++) {
            tvals.t += tvals.dt ;
            uchange = 1.0 ;
            itnum = 0 ;
            for(j=0;j<150;j++) {
                if(uchange<tol)
                    break ;
                assemble (nvar,0,0,elcode,itnum) ;

                solve(nvar,2,0) ;
            itnum += 1 ;

            }

            maxnodep = nodep = gp.N ;
            for(j=0;j<N.nodes;j++) {
                hnum = 1 ;
                if(nodep->x[0]==200000)
                    hnum = j ;
                if(fabs(nodep->u[1]) > fabs(maxnodep->u[1]) )
                    maxnodep = nodep ;
                nodep = nodep->nextnp ;
            }

            fprintf(fptr,"%d\t%g\t%g\t%g\n",i+1,tvals.t,
                gp.iptrs[hnum]->u[1], (gp.iptrs[hnum]->u[0]
                /gp.iptrs[hnum]->p[2])) ;

            if ((i+1)%tvals.iter == 0) {
                DrawElev();
                printf("Time step=%d,\tt=%fs\tIterations=%d\t
                    uchange=%g\n",i+1,tvals.t,itnum,uchange);
            }

                    tvals.dt = tvals.dt * tvals.dtfac ;
            set_ptou(0) ;
          }
      }
      return(0) ;
```

```
}

int       assemble(varnum,code,nset,elcode,itnum)
int       varnum, code, nset, elcode, itnum  ;
{
     double                   *K_init(), *L_init(),*F_init() ;
     struct fxnodes           *fn_init();
     struct element           *elp, tel ;
     struct belement          *belp, tbel ;
     double                        *p ;
     int                           i,j,ns,ntf,istart,iend,ind ;
     long                     ij ;
     char                                    m1[255], m2[255], m3[255], m4[255] ;
     FILE     *fptr, *put_fptr() ;

     istart = 0 ;
     iend = N.elms ;
     if(N.trans == 0)
          tvals.dt = 1.0 ;


     if(code< 2) {


          if(eqnsets[nset].Kp != NULL)
               free(eqnsets[nset].Kp) ;

          if( (eqnsets[nset].Kp = K_init(varnum,&eqnsets[nset].diagp,code)) == NULL )
               return(-1) ;

     if(N.sym != 0) {
               if(eqnsets[nset].Lp != NULL )
                    free(eqnsets[nset].Lp) ;
               if( (eqnsets[nset].Lp = L_init(varnum,&eqnsets[nset].diagp,code)) == NULL )
                    return(-1) ;
          }

          eqnsets[nset].neqns = Nukns ;
          if(eqnsets[nset].Fp != NULL) {
               free(eqnsets[nset].Fp) ;
          }
          if( ( eqnsets[nset].Fp = F_init(Nukns) )  == NULL  )
               return(-1) ;
     }
     else if (code == 2) {
          ind = 0 ;
          for(i=0;i<N.nodes;i++) {
               gp.iptrs[i]->ui = ind ;
               ind++ ;
          }
     }
     if(code < 1) {

          if( (ep.n = ep_init() ) == 0 )
               return(-1) ;
     }
     for(i=0;i<eqnsets[nset].neqns;i++)
          *(eqnsets[nset].Fp + i) = 0.0 ;
     elp = gp.El ;
     for(i=istart;i<iend;i++) {
          ns = get_PGKe(elp,&tel,varnum,elcode,&ntf,itnum) ;
     put_Ke(ns,ntf,&tel,varnum,eqnsets[nset].Kp,eqnsets[nset].Lp,eqnsets[nset].Fp,eqnsets[nset].diagp,code);
          elp = elp->nextelp ;
     }

     if(code < 2){

          EndMes() ;
     }

     return(0) ;
 }
 double    *K_init(nvar,thediagp,code)
 int       nvar, code ;
 long           *(*thediagp) ;
 {
     int  nj, i, j, nf , ind, n;
     double *kp, *dp ;
     unsigned      elsize ;
     long      bigint, nelem,*diagp, il ;
```

```
    if(nvar<0)
        n = N.vars ;
    else
        n = 1 ;
    nelem = N.nodes * n ;
    Nukns = nelem ;
    if(*thediagp != NULL)
        free(*thediagp) ;
    elsize = sizeof(long) ;
    if((*thediagp = (long int *) calloc((int)nelem,sizeof(long) )) == NULL)
        return(NULL) ;
    diagp = *thediagp ;
    *diagp = 0 ;
    ind = 0 ;
    gp.iptrs[0]->ui = ind ;
    diagp++ ;
    ind++ ;
    for(i=1;i<Nukns;i++) {
        gp.iptrs[i/n]->ui = ind ;
        ind++ ;
        if(code == 0) {
            if(ind < BandWidth)
                nf = ind ;
            else
                nf = BandWidth ;
        }
        else if(code == -1)    /*for assembling a diagonal matrix only*/
            nf = 1 ;
        else if(code == 2) {/*for assembling a pentadiagonal matrix*/
            if(ind == 1)
                nf = 1;
            else if(ind == 2)
                nf = 2 ;
            else
                nf = 3 ;
        }
        else
            nf = 2 * n;
        *diagp = *(diagp-1) + nf ;
        diagp++ ;
    }
    nelem = *(diagp-1) + 1 ;
    elsize = sizeof(double) ;
    bigint =  nelem * (long) elsize  ;
    kp =  dp = (double *) malloc(bigint)  ;
    for(il=0;il<nelem;il++)
        *(dp++) = 0.0 ;
    return(kp) ;
}
double   *L_init(nvar,thediagp,code)
int       nvar,code ;
long          *(*thediagp) ;
{
    int  nj, i, j, nf , ind;
    double *kp, *dp ;
    unsigned  elsize ;
    long      bigint, nelem, il, *diagp ;

    diagp = *thediagp ;
    for(i=0;i<Nukns;i++)
            diagp++ ;

    nelem = *(diagp-1) + 1 ;
    elsize = sizeof(double) ;
    bigint =  nelem * (long) elsize  ;
    kp =  dp = (double *) malloc(bigint)  ;
    for(il=0;il<nelem;il++)
        *(dp++) = 0.0 ;
    return(kp) ;
}
double    *F_init(num)
int       num ;
{
    double    *p ;
    unsigned       nelem, elsize ;

    nelem = num ;
    elsize = sizeof(double) ;
    p = (double *) calloc(nelem,elsize) ;
    return(p) ;
}
```

```
int       ep_init()
{
     int  i, ns, nsm ;
     struct element     *elp ;

     if(ep.K != NULL)
          free(ep.K) ;
     elp = gp.El ;
     nsm = 0 ;

     for(i=0;i<N.elms;i++) {
          nsm = ( (ns = nsf(elp->vtype)) > nsm ) ? ns : nsm ;
          elp = elp->nextelp ;
     }
     nsm *= N.vars ;
     if( (ep.K = (double *) calloc(nsm*nsm, sizeof(double)) ) == NULL )
          return(0) ;
     if(ep.F != NULL)
          free(ep.F) ;
     if( (ep.F = (double *) calloc(nsm, sizeof(double)) ) == NULL )
          return(0) ;
     if(N.trans > 0) {
          if(ep.S != NULL)
               free(ep.S) ;
          if( (ep.S = (double *) calloc(nsm*nsm, sizeof(double)) ) == NULL )
               return(0) ;
     }
     if(N.trans > 1) {
          if(ep.M != NULL)
               free(ep.M) ;
          if( (ep.M = (double *) calloc(nsm*nsm, sizeof(double)) ) == NULL )
          return(0) ;
     }
     if(N.nonlin > 0) {
          if(ep.J != NULL)
               free(ep.J) ;
          if( (ep.J = (double *) calloc(nsm*nsm, sizeof(double)) ) == NULL )
          return(0) ;
     }
     return(nsm) ;
}
int       put_Ke(ns,nt,elmntp,nvar,Kp,Lp,Fp,diagp,code)
int       ns, code ;
double    *Kp,  *Lp, *Fp ;
struct element *elmntp ;
long      *diagp ;
{
     int i, in, nvars, j, ii, jj, jn, frow, fcol, row, col, tj, isv ;
     struct node  *inp, *jnp ;
     struct fxnodes *the_fnp ;
     if(nvar < 0)
          nvars = N.vars ;
     else
          nvars = 1 ;

     for(i=0;i<nt;i++) {
          inp = elmntp->nps[i] ;
          frow = inp->i * nvars ;
          for(j=0;j<ns;j++) {
               jnp = elmntp->nps[j] ;
               fcol = jnp->i * nvars;
               for(ii=0;ii<nvars;ii++) {
                    row = frow + ii ;
                    for(jj=0;jj<nvars;jj++) {
                         col = fcol + jj ;

                         if( (col == row) && ( code == -1 ) ) {
                              if(N.trans == 0)
                                   *(Kp + row ) += KE(i*nvars+ii, j*nvars+jj) ;
                              else
                                   *(Kp + row ) += SE(i*nvars+ii, j*nvars+jj) ;
                         }
                         if(  (code == 2 ) || code == -1 ) {
                              *(Fp  + row ) += - KE(i*nvars+ii, j*nvars+jj) * jnp->u[jj] ;
                         }
                    }
               }
          }
          for(ii=0;ii<nvars;ii++) {
               row = frow + ii ;
               *(Fp + row) += FE(i*nvars+ii) ;
```

```
        }
    }
}
int       mpd_i(ind,nvar)
int       ind , nvar;
{
    int  i,j,n,mpd,td ;
    struct element  *elmntp ;

    mpd = 0 ;

    if(N.dims == 1)
        return(3) ;
    elmntp = gp.El ;
    for(i=0;i<N.elms;i++) {
        n = -1 ;
        for(j=0;j<elmntp->nnds;j++) {
            if(elmntp->nps[j] != NULL) {
                if(gp.iptrs[ind]->n == (elmntp->nps[j])->n )
                    n = gp.iptrs[ind]->n ;
            }
        }
        if( n >= 0 ) {
            for(j=0;j<elmntp->nnds;j++) {
                if(elmntp->nps[j] != NULL) {
                    if((elmntp->nps[j])->n >= 0)
                        td = d_i(ind,(elmntp->nps[j])->i) ;
                    mpd = (mpd > td ) ? mpd : td ;
                }
            }
        }
        elmntp = elmntp->nextelp ;
    }
    return(mpd) ;
}
int       d_i(n1,n2)
int       n1, n2 ;
{
    return(n1 - n2) ;
}

int       get_PGKe(elmntp,theElp,nvar,code,ntf,itnum)
struct element           *elmntp, *theElp ;
int       nvar, code, *ntf, itnum ;
{
    struct gausspts    g[MGPTS] ;
    struct shapefuncs      bf, tf, *fgp;
    struct node            *np[MNSF] ;
    struct govfuncs        gbf, gtf ;
    int                    i, j, nbf, ni, nm, dir, n, jj ;
    double                 tdt, mtdt, sdiag ;
    double                 *fptr ;
    if(nvar<0)
        n = N.vars ;
    else
        n = 1 ;
    initm(ep.K,ep.n*ep.n) ;
    if(N.trans > 0)
        initm(ep.S,ep.n*ep.n) ;
    if(N.nonlin > 0)
        initm(ep.J,ep.n*ep.n) ;
    initm(ep.F,ep.n) ;

    ElPick(elmntp,theElp,ntf,&dir) ;
    nbf = theElp->nnds ;

    if(code > 0 || elmntp->matrices == NULL || N.nonlin > 0) {
        ni = get_gspts(theElp->vtype,g) ;
        for(j=0;j<theElp->nnds;j++)
            np[j] = theElp->nps[j];

        for(i=0;i<ni;i++) {
            get_shape(theElp->vtype,&g[i],&bf) ;
            get_shape(theElp->gtype,&g[i],&tf) ;
            get_PGgf(theElp,&gbf,&gtf,&bf,&tf,np,g[i].w) ;
            update_PGKe(&gbf,&gtf,nbf,*ntf,np,nvar,theElp) ;
        }
    }
    if(itnum == 0 && N.nonlin > 0) {
        if(elmntp->matrices != NULL)
            free(elmntp->matrices) ;
        nm = *ntf *n ;
        if((elmntp->matrices = (double *)calloc(nm ,sizeof(double)) ) == NULL)
            return(-1) ;
        fptr = elmntp->matrices ;
        for(i=0;i<*ntf*n;i++){
            *fptr = FE(i);
            fptr++ ;
        }
    }
    else {
        fptr = elmntp->matrices ;
        for(i=0;i<*ntf*n;i++) {
            FE(i) = *fptr ;
            fptr++ ;
        }
    }
    if(N.nonlin > 0) {
        for(i=0;i<(*ntf)*n;i++) {
            for(j=0;j<nbf;j++) {
                for(jj=0;jj<n;jj++) {
                    FE(i) -= (SE(i,j*n+jj) * (theElp->nps[j])->u[jj]) ;
                    SE(i,j*n+jj) += tdt * JE(i,j*n+jj) ;
                }
            }
        }
    }
    return(nbf) ;
}
int         ElPick(elmntp,theElp,ntfp,dirp)
struct element      *elmntp, *theElp ;
int         *ntfp, *dirp ;
{
    int  j;

    *dirp = 1 ;
    *ntfp = nsf(elmntp->gtype) ;
    *theElp = *elmntp ;
    return(0) ;
}
int         get_PGgf(elp,gbfp,gtfp,bfp,tfp,np,w)
struct element              *elp ;
struct shapefuncs           *bfp, *tfp ;
struct node                 *np[] ;
struct govfuncs             *gbfp, *gtfp ;
double                      w ;
{
    double    J[NDIMS][NDIMS], get_J() ;
    int       i, j ;

    gbfp->dof = bfp->dof ;
    gtfp->dof = tfp->dof ;

    for(i=0;i<N.params;i++) {
        gbfp->p[i] = elp->p[i] ;
    }
            i = (gbfp->dof > gtfp->dof) ? gbfp->dof : gtfp->dof ;
            for(j=0;j<i;j++) {
                gbfp->f[j] = 0.0  ;
                gtfp->f[j] = 0.0  ;
                gbfp->dfdx[j] = 0.0 ;
                gtfp->dfdx[j] = 0.0 ;
            }
    for(j=0;j<bfp->dof;j++)
        gbfp->f[j] = bfp->f[j]  ;
    for(j=0;j<tfp->dof;j++)
        gtfp->f[j] = tfp->f[j]  ;

    gbfp->w = w ;
    return(0) ;
}
```

fecom.c

```c
#include "fe.h"
#define  KE(A,B)  ( *(ep.K + ep.n*(A) + (B)) )
#define  SE(A,B)  ( *(ep.S + ep.n*(A) + (B)) )
#define  ME(A,B)  ( *(ep.M + ep.n*(A) + (B)) )
#define  JE(A,B)  ( *(ep.J + ep.n*(A) + (B)) )
#define  FE(B)  ( *(ep.F + B) )
extern struct control  N ;
extern struct pointers gp;
extern struct RegMesh      Mesh ;
extern struct fxnodes   *first_fnp ;
extern int          Nukns ;
extern struct elmpointers ep ;
extern struct transient          tvals ;
extern struct eqnset    eqnsets[4] ;
extern double                 uchange ;

int      solve(nvar,code,nset)
int      code, nvar, nset ;
{
    int  i, j, ind, solvecode ;
    double   *dp, varsum ;

    if(code > 1)
        solvecode = 1 ;
    else
        solvecode = code ;
    if(code < 0) {
        solvecode = 2 ;
        code = - code ;
    }
/*
    printf(" Now solving ...") ;
*/
    if((N.sym ==  0)  ||  (code == 3))
        actcol(eqnsets[nset].Kp,eqnsets[nset].Fp,eqnsets[nset].diagp, eqnsets[nset].neqns, solvecode) ;
    else
        uactcl(eqnsets[nset].Kp,eqnsets[nset].Lp,eqnsets[nset].Fp,eqnsets[nset].diagp, eqnsets[nset].neqns,
            solvecode) ;
/*
    printf(" All done !\n") ;
*/

    dp = eqnsets[nset].Fp ;
    if((code > 0) && (code != 3)) {
        if(nvar >= 0) {
            ind = 0 ;
            uchange = 0.0 ;
            for(i=0;i<N.nodes;i++) {
                if(code == 1) {
                    uchange += fabs(gp.iptrs[i]->u[nvar] - *dp) ;
                    gp.iptrs[i]->u[nvar] = *dp ;
                }
                if(code == 2) {
                    *(eqnsets[nset+1].Fp + ind) = *(eqnsets[nset].Fp + gp.iptrs[i]->l) ;
                /*
                    gp.iptrs[i]->p[4] = *(eqnsets[nset].Fp + gp.iptrs[i]->l) ;
                */
                }
                if(code == 4) {
                /*
                    printf(" %f  %f\n",gp.iptrs[i]->u[nvar], *dp);
                */
                    gp.iptrs[i]->u[nvar] += *dp ;
                }
                dp++;
                ind++ ;
            }
        }
        else {
            uchange = 0.0 ;
            varsum = 0.0 ;
            for(i=0;i<N.nodes;i++) {
                for(j=0;j<N.vars;j++) {
                    if(code == 1)
                        gp.iptrs[i]->u[j] = *dp;
                    if(code == 2) {
                        uchange += *dp * *dp;
                        gp.iptrs[i]->u[j] += *dp ;
                        varsum += gp.iptrs[i]->u[j] * gp.iptrs[i]->u[j] ;
                    }
                    dp++;
                }
            }
        uchange = sqrt (uchange/varsum) ;
        }
    }
    return(0) ;
}
int     actcol(Kp,Fp,diagp,neqns,code)
double  *Kp, *Fp ;
long    *diagp ;
int     neqns, code ;
{
    int i, j, nf, id, n, ni, nj ;
    double  dot(), *ip, *idp, t ;
    char    m1[255], m2[255], m3[255], m4[255] ;

/*
    if(code < 2) {
        sprintf(m1," Now factoring equation # ") ;
        sprintf(m2," 0 ") ;
        sprintf(m3," ") ;
        sprintf(m4," ") ;
        StartMes(m1,m2,m3,m4) ;
    }
*/
    for(j=1;j<neqns;j++) {
/*
        if( (j % 10 == 0) && (code < 2)) {
            sprintf(m2,"  %d ",j) ;
            UpDateMes(m1,m2,m3,m4) ;
        }
*/
        nf = *(diagp+j) - *(diagp+j-1) ;
        if(code < 2) {
            ip = Kp + *(diagp+j-1) + 1 ;
            id = j - nf + 1 ;

            for(i=1;i<nf;i++) {
                n = ((nj = i-1) < (ni = *(diagp+id) - *(diagp+id-1) - 1)) ? nj : ni ;
                idp = Kp + *(diagp+id++) ;
                *ip++ -= dot(ip-n-1,idp-n,n) ;
            }
            ip = Kp + *(diagp+j-1) + 1 ;
            id = j - nf + 1 ;
            idp = Kp + *(diagp+j) ;
            for(i=1;i<nf;i++) {
                t = *ip ;
                *ip /= *(Kp + *(diagp+id++)) ;
                *idp -= t * *ip++ ;
            }
        }
        ip = Kp + *(diagp+j-1) + 1 ;
        id = j - nf + 1 ;
        if(code > 0)
            *(Fp + j) -= dot(ip,Fp+id,nf-1) ;
    }
    if(code == 0) {
        EndMes() ;
        return(0) ;
    }
    for(j=0;j<neqns;j++)
        *(Fp + j) /= *(Kp + *(diagp+j)) ;

    for(j=neqns-1;j>-1;j--) {
        if(j == 0)
            nf = 1;
        else
            nf = *(diagp+j) - *(diagp+j-1) ;
        for(i=1;i<nf;i++) {
            *(Fp + j - nf + i) -= *(Fp + j) * *(Kp + *(diagp+j-1) + i) ;
        }
    }
    if(code < 2)
        EndMes() ;
    return(0) ;
}
```

```
int       uactcl(Kp,Lp,Fp,diagp,neqns,code)
double    *Kp, *Lp, *Fp ;
long      *diagp ;
int       neqns, code ;
{
     int  i, j, nf, id, n, ni, nj ;
     double    dot(), *ip, *iLp, *idp, t ;
     char      m1[255], m2[255], m3[255], m4[255] ;

/*   if(code < 2) {
          sprintf(m1," Now factoring equation # ") ;
          sprintf(m2," 0 ") ;
          sprintf(m3," ") ;
          sprintf(m4," ") ;
          StartMes(m1,m2,m3,m4) ;
     }
*/
     for(j=1;j<neqns;j++) {
/*
          if( (j % 10 == 0) && (code < 2)) {
               sprintf(m2,"  %d ",j) ;
               UpDateMes(m1,m2,m3,m4) ;
          }
*/
          nf = *(diagp+j) - *(diagp+j-1) ;
          if(code < 2) {
               iLp = Lp + *(diagp+j-1) + 1 ;
               id = j - nf + 1 ;

               for(i=0;i<nf-1;i++) {
                    n = ((nj = i) < (ni = *(diagp+id) - *(diagp+id-1) - 1)) ? nj : ni ;
                    idp = Kp + *(diagp+id) ;
                    id++ ;
                    *iLp = (*iLp - dot(iLp-n,idp-n,n))/ *idp ;
                    iLp++ ;
               }
               ip = Kp + *(diagp+j-1) + 1 ;
               id = j - nf + 1 ;

               for(i=0;i<nf;i++) {
                    n = ((nj = i) < (ni = *(diagp+id) - *(diagp+id-1) - 1)) ? nj : ni ;
                    idp = Lp + *(diagp+id) ;
                    id++ ;
                    *ip -= dot(ip-n,idp-n,n) ;
                    ip++ ;
               }
          }
          iLp = Lp + *(diagp+j-1) + 1 ;
          id = j - nf + 1 ;
          if(code > 0)
               *(Fp + j) -= dot(iLp,Fp+id,nf-1) ;
     }
     if(code == 0) {
          EndMes() ;
          return(0) ;
     }
     for(j=neqns-1;j>-1;j--) {
          *(Fp + j) /= *(Kp + *(diagp+j)) ;
          if(j == 0)
               nf = 1;
          else
               nf = *(diagp+j) - *(diagp+j-1) ;
          for(i=1;i<nf;i++) {
               *(Fp + j - nf + i) -= *(Fp + j) * *(Kp + *(diagp+j-1) + i) ;
          }
     }
     if(code < 2)
          EndMes() ;
     return(0) ;
}
double    dot(p1,p2,n)
double    *p1, *p2 ;
int       n ;
{
     register double     t=0.0 ;

     while(n-- > 0) {
               t += *p1++ * *p2++ ;
     }
     return(t) ;
}
```

```
int       initm(K,n)
double    *K ;
int       n ;
{
     register int  i ;
     for(i=0;i<n;i++)
               *(K++) = 0.0 ;
     return(0) ;
}
```

fegauss.c

```
#include "fe.h"
int      get_gspts(i_type,g)
int      i_type ;
struct gausspts          g[] ;
{
    int n ;

    switch (i_type) {

        case 1 :
            n = gs_pt(g) ;
            break ;

        default :
            n = gs_1D1(g) ;
            break ;

    }
    return(n) ;
}
int      gs_pt(g)
struct gausspts          g[] ;
{
    g[0].x =  0.0 ;
    g[0].y =  0.0 ;
    g[0].z =  0.0 ;
    g[0].w =  1.0 ;

    return(1) ;
}
int      gs_1D1(g)
struct gausspts          g[] ;
{
    g[0].x =  0.0 ;
    g[0].y =  0.0 ;
    g[0].z =  0.0 ;
    g[0].w =  2.0 ;

    return(1) ;
}
```

festuff.c

```
#include "fe.h"
extern struct control  N ;
extern double      UW ;
int  get_shape(el_type,x,psf)
int  el_type ;
struct gausspts         *x ;
struct shapefuncs       *psf ;
{
    int  err ;

    err = shap_lsl(x,psf) ;

    return(err) ;
}
int  nsf(el_type)
int  el_type ;
{
    struct gausspts          x ;
    struct shapefuncs        psf ;
    int  err ;
    x.x = 100.0 ;
    x.y = 100.0 ;
    x.z = 100.0 ;
    err = get_shape(el_type,&x,&psf) ;
    return(err) ;
}
int  nbounds(el_type)
int  el_type ;
{
    int  err ;

    err = 2 ;
    return(err) ;
}
int       get_nlcs(el_type,k,r,s)
int       k, el_type ;
double    *r, *s ;
{

            switch (k) {
                case 0 :
                    *r = 1.0 ;
                    *s = 1.0 ;
                    break ;
                case 1 :
                    *r = -1.0 ;
                    *s = 1.0 ;
                    break ;
            }
            return(2) ;

}
int  shap_lsl(x,p)
struct gausspts         *x ;
struct shapefuncs       *p ;
{
    double   r,mr,pr,t ;

    r = x->x ;
    if( (r<-1.0) || (r>1.0) ) return(2) ;

    t = 0.5 ;
    mr = 1.0 - r ;
    pr = 1.0 + r ;

    p->dof = 2 ;

    p->f[0] =  t * mr ;
    p->f[1] =  t * pr ;

    if(x->w < 0.0)
        return(0) ;
    p->dfdr[0] = -0.5 ;
    p->dfdr[1] =  0.5 ;

    return(0) ;
}
int shap_pt(x,p)
struct gausspts          *x ;
struct shapefuncs        *p ;
{
    double   r;

    r = x->x ;
    if( (r<-1.0) || (r>1.0) ) return(1) ;

    p->dof = 1 ;
    p->f[0] =  1 ;
    p->dfdr[0] = 0.0 ;

    return(0) ;
}
```

feio.c

(see Appendix C)

fetest.c

(see Appendix C)

feice.c

(see Appendix C)

fewindow.c

(see Appendix C)

APPENDIX B - IMPLEMENTATION OF THE FEM FOR THE ST. VENANT EQUATIONS

B.1 The Conservation Form

B.1.1 Introduction

Given the conservation form of the St. Venant Equations,

$$\frac{\partial A}{\partial t} + \frac{\partial Q}{\partial x} = 0 \qquad [2.6]$$

$$\frac{\partial Q}{\partial t} + \frac{\partial QU}{\partial x} + \frac{\partial}{\partial x}\left(\frac{gAH}{2}\right) - \frac{gAH}{2B}\frac{dB}{dx} = gA\,(S_o\text{-}S_f) \qquad [2.46]$$

The semi-discrete form of the equations may be written:

$$\int_L \left(\mathbf{v}_i \mathbf{f}_j \frac{dA_j}{dt} - \frac{dv_i}{dx}\mathbf{f}_j Q_j \right) dx + \left[\mathbf{v}_i\,(Q)\right]_0^L = \{0\} \qquad [B.1]$$

$$\int_L \left(\mathbf{v}_i \mathbf{f}_j \frac{dQ_j}{dt} - U\frac{dv_i}{dx}\mathbf{f}_j Q_j + \frac{gH}{2}\frac{dv_i}{dx}\mathbf{f}_j A_j - \frac{gH}{2B}\frac{dB}{dx}\mathbf{v}_i\mathbf{f}_j A_j - gS_o\,\mathbf{v}_i\mathbf{f}_j A_j + \frac{|U|}{RC_*^2}\mathbf{v}_i\mathbf{f}_j Q_j \right) dx$$

$$+ \left[\mathbf{v}_i\left(QU + \frac{gAH}{2}\right)\right]_0^L = \{0\} \qquad [B.2]$$

where integration by parts has been used. Also, we may define a new variable, *f*, such that:

$$f = \frac{|U|}{RC_*^2} \qquad [B.3]$$

Each elemental contribution may considered separately in terms of the unknown variables and the equation being considered...

$[\mathbf{S}_{aa}]$ ≡ mass matrix (continuity equation, area term)

$[\mathbf{S}_{aq}]$ ≡ mass matrix (continuity equation, discharge term)

$[\mathbf{S}_{qq}]$ ≡ mass matrix (momentum equation, discharge term)

$[\mathbf{S}_{qa}]$ ≡ mass matrix (momentum equation, area term)

$[\mathbf{K}_{aa}]$ ≡ stiffness matrix (continuity equation, area term)

$[\mathbf{K}_{aq}]$ ≡ stiffness matrix (continuity equation, discharge term)

$[\mathbf{K}_{qa}]$ ≡ stiffness matrix (momentum equation, area term)

$[\mathbf{K}_{qq}]$ ≡ stiffness matrix (momentum equation, discharge term)

where the first subscript denotes the equation (a - continuity or 'area equation', q - momentum or 'discharge equation'). The second subscript denote the variable (a - area, q - discharge).

B.1.2 The Bubnov-Galerkin Method

In the Bubnov-Galerkin scheme, the test functions are taken equal to the interpolation functions such that:

$$[\mathbf{S}_{aa}] = [\mathbf{S}_{qq}] \equiv \left[\int_e \mathbf{f}_i\, \mathbf{f}_j\, dx\right] \qquad [B.4]$$

$$[\mathbf{S}_{aq}] = [\mathbf{S}_{qa}] \equiv [0] \qquad [B.5]$$

$$[\mathbf{K}_{aa}] \equiv [0] \qquad [B.6]$$

$$[\mathbf{K}_{aq}] \equiv \left[\int_e -\frac{d\mathbf{f}_i}{dx}\, \mathbf{f}_j\, dx\right] \qquad [B.7]$$

$$[\mathbf{K}_{qa}] \equiv \left[\int_e \left(-\frac{gH}{2}\frac{d\mathbf{f}_i}{dx}\mathbf{f}_j + \frac{gH}{2B}\frac{dB}{dx}\mathbf{f}_i\mathbf{f}_j - gS_o\,\mathbf{f}_i\mathbf{f}_j\right)dx\right] \tag{B.8}$$

$$[\mathbf{K}_{qq}] \equiv \left[\int_e \left(-U\frac{d\mathbf{f}_i}{dx}\mathbf{f}_j + f\,\mathbf{f}_i\mathbf{f}_j\right)dx\right] \tag{B.9}$$

B.1.3 The Taylor-Galerkin Method

For this study, the Taylor-Galerkin method is modeled using a Bubnov-Galerkin formulation of an alternate system of equations. This was given in Chapter 3 as:

$$\frac{\partial\{\Phi\}}{\partial t} + \left([\mathbf{A}]\frac{\partial\{\Phi\}}{\partial x} + \{S\}\right)$$
$$-\frac{\Delta t}{2}\left(\frac{\partial\{S\}}{\partial\{\Phi\}}\left([\mathbf{A}]\frac{\partial\{\Phi\}}{\partial x} + \{S\}\right) + \frac{\partial}{\partial x}\left([\mathbf{A}]\left([\mathbf{A}]\frac{\partial\{\Phi\}}{\partial x} + \{S\}\right)\right)\right) = \{0\} \tag{3.74}$$

for the non-conservation form of the equations.

To avoid confusion with the unknown variable, A (area), let the convection matrix be described as:

$$[\mathbf{A}] \equiv \begin{bmatrix} a_{aa} & a_{aq} \\ a_{qa} & a_{qq} \end{bmatrix} \tag{B.10}$$

where,

$$a_{aa} = 0;\ a_{aq} = 1;\ a_{qa} = (gH - U^2);\ \text{and}\ a_{qq} = 2U \tag{B.11}$$

This system may then be expanded to obtain,

$$\frac{\partial}{\partial t}\begin{Bmatrix} A \\ Q \end{Bmatrix} + \left(\begin{bmatrix} a_{aa} & a_{aq} \\ a_{qa} & a_{qq} \end{bmatrix}\frac{\partial}{\partial x}\begin{Bmatrix} A \\ Q \end{Bmatrix} + \begin{Bmatrix} 0 \\ -gAS_o + f\,Q \end{Bmatrix}\right)$$
$$-\frac{\Delta t}{2}\left(\frac{\partial}{\partial\begin{Bmatrix} A \\ Q \end{Bmatrix}}\begin{Bmatrix} 0 \\ -gAS_o + f\,Q \end{Bmatrix}\left(\begin{bmatrix} a_{aa} & a_{aq} \\ a_{qa} & a_{qq} \end{bmatrix}\frac{\partial}{\partial x}\begin{Bmatrix} A \\ Q \end{Bmatrix} + \begin{Bmatrix} 0 \\ -gAS_o + f\,Q \end{Bmatrix}\right)\right)$$

$$-\frac{\Delta t}{2}\left(\frac{\partial}{\partial x}\left(\begin{bmatrix}a_{aa} & a_{aq}\\ a_{qa} & a_{qq}\end{bmatrix}\left(\begin{bmatrix}a_{aa} & a_{aq}\\ a_{qa} & a_{qq}\end{bmatrix}\frac{\partial}{\partial x}\begin{Bmatrix}A\\ Q\end{Bmatrix} + \begin{Bmatrix}0\\ -gAS_o + f\,Q\end{Bmatrix}\right)\right)\right) = \{0\} \qquad [B.12]$$

given that,

$$\{\Phi\} \equiv \begin{Bmatrix}A\\ Q\end{Bmatrix} \qquad [B.13]$$

and,

$$\{S\} \equiv \begin{Bmatrix}0\\ -gAS_o + f\,Q\end{Bmatrix} \qquad [B.14]$$

Note that the first row in equation [B.12] represents the original system of equations to be solved (in non-conservation form, for now). The second row includes the derivative of the source term, while the third row provides the second derivative with respect to x, which acts as apparent diffusion.

An extra term is included separately when channel width is a function of x, given that:

$$gA\frac{\partial H}{\partial x} = gH\frac{\partial A}{\partial x} - \frac{gAH}{B}\frac{dB}{dx} \qquad [B.15]$$

which may then be included as a source term such that,

$$\{S\} \equiv \begin{Bmatrix}0\\ -\left(gS_o + \frac{gH}{B}\frac{dB}{dx}\right)A + f\,Q\end{Bmatrix} \qquad [B.16]$$

If we make the approximation,

$$\frac{\partial(fQ)}{\partial A} \approx -\frac{2|Q|QB}{A^3C^2} \qquad [B.17]$$

then

$$\frac{\partial\{S\}}{\partial\{\Phi\}} = \frac{\partial}{\partial\begin{Bmatrix}A\\ Q\end{Bmatrix}}\begin{Bmatrix}0\\ -\left(gS_o + \frac{gH}{B}\frac{dB}{dx}\right)A + f\,Q\end{Bmatrix}$$

$$= \begin{bmatrix} 0 & 0 \\ -\left(gS_o + \frac{gH}{B}\frac{dB}{dx}\right) - \left(\frac{2fQ}{A}\right) & 2f \end{bmatrix} \qquad [B.18]$$

For simplicity call these,

$$\begin{bmatrix} b_{aa} & b_{aq} \\ b_{qa} & b_{qq} \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ -\left(gS_o + \frac{gH}{B}\frac{dB}{dx}\right) - \left(\frac{2fQ}{A}\right) & 2f \end{bmatrix} \qquad [B.19]$$

Considering then the second row in equation [B.12], substitution gives,

$$-\frac{\Delta t}{2}\left(\begin{bmatrix} b_{aa} & b_{aq} \\ b_{qa} & b_{qq} \end{bmatrix}\left(\begin{bmatrix} a_{aa} a_{aq} \\ a_{qa} a_{qq} \end{bmatrix}\frac{\partial}{\partial x}\left\{\begin{matrix} A \\ Q \end{matrix}\right\} + \left\{\begin{matrix} 0 \\ -\left(gS_o + \frac{gH}{B}\frac{dB}{dx}\right)A + fQ \end{matrix}\right\}\right)\right) \qquad [B.20]$$

multiplying through and simplifying, (noting that $a_{aa} = b_{aa} = b_{aq} = 0$ and $a_{aq} = 1$), this becomes:

$$-\frac{\Delta t}{2}\left\{\begin{matrix} 0 \\ b_{qq}\left(-\left(gS_o + \frac{gH}{B}\frac{dB}{dx}\right)A + a_{qa}\frac{\partial A}{\partial x}\right) + b_{qq} f Q + (b_{qa} + a_{qq}b_{qq})\frac{\partial Q}{\partial x} \end{matrix}\right\}$$

$$\equiv \left\{\begin{matrix} \text{contribution to continuity equation} \\ \text{contribution to momentum equation} \end{matrix}\right\} \qquad [B.21]$$

Since there are no source terms in the continuity equation (which depend on the solution variables A and Q), there is no contribution from this term of equation [B.12].

Consider next the third row of equation [B.12], multiplying through and simplifying yields,

$$-\frac{\Delta t}{2}\frac{\partial}{\partial x}\left\{\begin{matrix} -\left(gS_o + \frac{gH}{B}\frac{dB}{dx}\right)A + a_{qa}\frac{\partial A}{\partial x} + fQ + a_{qq}\frac{\partial Q}{\partial x} \\ a_{qq}\left(-\left(gS_o + \frac{gH}{B}\frac{\partial B}{\partial x}\right)A\right) + a_{qa}a_{qq}\frac{\partial A}{\partial x} + a_{qq} f Q + \left(a_{qa}a_{aq} + a_{qq}^2\right)\frac{\partial Q}{\partial x} \end{matrix}\right\}$$

$$\equiv \left\{ \begin{array}{l} \text{contribution to continuity equation} \\ \text{contribution to momentum equation} \end{array} \right\} \tag{B.22}$$

The mass matrices will remain the same as those used in the Bubnov-Galerkin method (equations [B.4] and [B.5]). The stiffness matrices become:

$$[\mathbf{K}_{aa}] \equiv \left[\int_e \frac{\Delta t}{2} \left(- \left(gS_o + \frac{gH}{B} \frac{dB}{dx} \right) \frac{d\mathbf{f}_i}{dx} \mathbf{f}_j + a_{qa} \frac{d\mathbf{f}_i}{dx} \frac{d\mathbf{f}_j}{dx} \right) dx \right] \tag{B.23}$$

$$[\mathbf{K}_{aq}] \equiv \left[\int_e \left(- \frac{d\mathbf{f}_i}{dx} \mathbf{f}_j + \frac{\Delta t}{2} \left(f \frac{d\mathbf{f}_i}{dx} \mathbf{f}_j + a_{qq} \frac{d\mathbf{f}_i}{dx} \frac{d\mathbf{f}_j}{dx} \right) \right) dx \right] \tag{B.24}$$

$$[\mathbf{K}_{qa}] \equiv \left[\int_e \left(\begin{array}{c} \left(- \frac{gH}{2} \frac{d\mathbf{f}_i}{dx} \mathbf{f}_j - \left(gS_o + \frac{gH}{2B} \frac{dB}{dx} \right) \mathbf{f}_i \mathbf{f}_j \right) \\ + \frac{\Delta t}{2} b_{qq} \left(- \left(gS_o + \frac{gH}{B} \frac{dB}{dx} \right) \mathbf{f}_i \mathbf{f}_j + a_{qa} \mathbf{f}_i \frac{d\mathbf{f}_j}{dx} \right) \\ + \frac{\Delta t}{2} \left(a_{qq} \left(- \left(gS_o + \frac{gH}{B} \frac{dB}{dx} \right) \frac{d\mathbf{f}_i}{dx} \mathbf{f}_j \right) + a_{qa} a_{qq} \frac{d\mathbf{f}_i}{dx} \frac{d\mathbf{f}_j}{dx} \right) \end{array} \right) dx \right] \tag{B.25}$$

$$[\mathbf{K}_{qq}] \equiv \left[\int_e \left(\begin{array}{c} \left(- U \frac{d\mathbf{f}_i}{dx} \mathbf{f}_j + f \mathbf{f}_i \mathbf{f}_j \right) \\ + \frac{\Delta t}{2} \left(b_{qq} f \mathbf{f}_i \mathbf{f}_j + \left(b_{qa} + a_{qq} b_{qq} \right) \mathbf{f}_i \frac{d\mathbf{f}_j}{dx} \right) \\ + \frac{\Delta t}{2} \left(a_{qq} f \frac{d\mathbf{f}_i}{dx} \mathbf{f}_j + \left(a_{qa} a_{aq} + a_{qq}^2 \right) \frac{d\mathbf{f}_i}{dx} \frac{d\mathbf{f}_j}{dx} \right) \end{array} \right) dx \right] \tag{B.26}$$

where the terms associated with the original system have been represented as before in the Bubnov-Galerkin approach (equations [B.4] - [B.9]) and therefore represent the conservation form of the governing equations. The additional terms, associated with the implementation of the Taylor-Galerkin scheme have been derived using the non-conservation form of the equations.

B.1.4 The Petrov-Galerkin Method

The Petrov-Galerkin method is modeled using a Bubnov-Galerkin formulation of the conservation form of equations in conjunction with an upwinding contribution. As presented in Chapter 3 the method is implemented as:

$$\begin{bmatrix} \mathbf{f}_i & 0 \\ 0 & \mathbf{f}_i \end{bmatrix} \begin{Bmatrix} C \\ M \end{Bmatrix} + \omega \frac{\Delta x}{2} [\mathbf{W}] \frac{d}{dx} \begin{bmatrix} \mathbf{f}_i & 0 \\ 0 & \mathbf{f}_i \end{bmatrix} \begin{Bmatrix} C \\ M \end{Bmatrix} = \{0\} \qquad [3.79]$$

If we write:

$$[\mathbf{W}] \equiv \begin{bmatrix} w_{aa} & w_{aq} \\ w_{qa} & w_{qq} \end{bmatrix} \qquad [B.27]$$

then the mass matrices become,

$$[S_{aa}] \equiv \left[\int_e \left\{ (\mathbf{f}_i\ \mathbf{f}_j) + \left(w_{aa}\, \omega \frac{\Delta x}{2} \frac{d\mathbf{f}_i}{dx}\, \mathbf{f}_j \right) \right\} dx \right] \qquad [B.28]$$

$$[S_{aq}] \equiv \left[\int_e \left(w_{aq}\, \omega \frac{\Delta x}{2} \frac{d\mathbf{f}_i}{dx}\, \mathbf{f}_j \right) dx \right] \qquad [B.29]$$

$$[S_{qq}] \equiv \left[\int_e \left\{ (\mathbf{f}_i\ \mathbf{f}_j) + \left(w_{qq}\, \omega \frac{\Delta x}{2} \frac{d\mathbf{f}_i}{dx}\, \mathbf{f}_j \right) \right\} dx \right] \qquad [B.30]$$

$$[S_{qa}] \equiv \left[\int_e \left(w_{qa}\, \omega \frac{\Delta x}{2} \frac{d\mathbf{f}_i}{dx}\, \mathbf{f}_j \right) dx \right] \qquad [B.31]$$

The stiffness matrices become:

$$[\mathbf{K}_{aa}] \equiv \left[\int_e \omega \frac{\Delta x}{2} w_{aq} \left(- \left(gS_o + \frac{gH}{B} \frac{dB}{dx} \right) \frac{d\mathbf{f}_i}{dx}\, \mathbf{f}_j + a_{qa} \frac{d\mathbf{f}_i}{dx} \frac{d\mathbf{f}_j}{dx} \right) dx \right] \qquad [B.32]$$

$$[\mathbf{K}_{aq}] \equiv \left[\int_e \left(- \frac{d\mathbf{f}_i}{dx}\, \mathbf{f}_j + \omega \frac{\Delta x}{2} \left(w_{aa} \frac{d\mathbf{f}_i}{dx} \frac{d\mathbf{f}_j}{dx} + w_{aq}\, f \frac{d\mathbf{f}_i}{dx}\, \mathbf{f}_j + w_{aq}\, a_{qq} \frac{d\mathbf{f}_i}{dx} \frac{d\mathbf{f}_j}{dx} \right) \right) dx \right] \qquad [B.33]$$

$$[\mathbf{K}_{qa}] \equiv \left[\int_e \begin{pmatrix} \left(-\frac{gH}{2}\frac{d\mathbf{f}_i}{dx}\,\mathbf{f}_j - \left(gS_o + \frac{gH}{2B}\frac{dB}{dx} \right) \mathbf{f}_i\,\mathbf{f}_j \right) \\ + \omega\,\frac{\Delta x}{2}\, w_{qq} \left(-\left(gS_o + \frac{gH}{B}\frac{dB}{dx} \right) \frac{d\mathbf{f}_i}{dx}\,\mathbf{f}_j + a_{qa}\,\frac{d\mathbf{f}_i}{dx}\,\frac{d\mathbf{f}_j}{dx} \right) \end{pmatrix} dx \right] \qquad [B.34]$$

$$[\mathbf{K}_{qq}] \equiv \left[\int_e \begin{pmatrix} \left(U\,\frac{d\mathbf{f}_i}{dx}\,\mathbf{f}_j + f\,\mathbf{f}_i\mathbf{f}_j \right) \\ + \ \omega\,\frac{\Delta x}{2} \left(w_{qq}\,f\,\frac{d\mathbf{f}_i}{dx}\,\mathbf{f}_j \ + \ (w_{qa} + w_{qq}a_{qq})\,\frac{d\mathbf{f}_i}{dx}\,\frac{d\mathbf{f}_j}{dx} \right) \end{pmatrix} dx \right] \qquad [B.35]$$

Note that the Dissipative-Galerkin method requires:

$$[\mathbf{W}] \equiv \begin{bmatrix} w_{aa} & w_{aq} \\ w_{qa} & w_{qq} \end{bmatrix} \equiv \begin{bmatrix} 0 & \frac{1}{|U + c|} \\ \frac{\left(c^2 - U^2\right)}{|U + c|} & \frac{2U}{|U + c|} \end{bmatrix} \qquad [B.36]$$

while for the Characteristic-Dissipative-Galerkin method:

$$[\mathbf{W}] \equiv \begin{bmatrix} w_{aa} & w_{aq} \\ w_{qa} & w_{qq} \end{bmatrix} \equiv \frac{1}{2c} \begin{bmatrix} \frac{\left(c^2 - U^2\right)}{|U + c| - |U - c|} & \frac{(U + c)}{|U + c|} - \frac{(U - c)}{|U - c|} \\ \left(c^2 - U^2\right) \left(\frac{(U + c)}{|U + c|} - \frac{(U - c)}{|U - c|} \right) & \frac{(U + c)^2}{|U + c|} - \frac{(U - c)^2}{|U - c|} \end{bmatrix} \qquad [B.37]$$

B.1.5 The Least-Squares Method

This requires implementation equivalent to a Bubnov-Galerkin formulation of:

$$\begin{aligned} &\frac{\partial\{\Phi\}}{\partial t} + [\mathbf{A}]\,\frac{\partial\{\Phi\}}{\partial x} + \{S\} \\ &\qquad - \Delta t\,\theta \left(\frac{\partial[\mathbf{A}]}{\partial x} + \frac{\partial\{S\}}{\partial\{\Phi\}} \right) \left(\frac{\partial\{\Phi\}}{\partial t} + [\mathbf{A}]\,\frac{\partial\{\Phi\}}{\partial x} + \{S\} \right) \\ &\qquad\qquad - \Delta t\,\theta\,\frac{\partial}{\partial x} \left([\mathbf{A}]^{\mathrm{T}} \left(\frac{\partial\{\Phi\}}{\partial t} + [\mathbf{A}]\,\frac{\partial\{\Phi\}}{\partial x} + \{S\} \right) \right) = \{0\} \end{aligned} \qquad [3.102]$$

The third line of this equation is analogous to a Petrov-Galerkin method using,

$$[W] \equiv \frac{2\,\theta\,\Delta t}{\omega\,\Delta x}\,[A]^T \equiv \frac{2\,\theta\,\Delta t}{\omega\,\Delta x}\begin{bmatrix} a_{aa} & a_{qa} \\ a_{aq} & a_{qq} \end{bmatrix} \equiv \frac{2\,\theta\,\Delta t}{\omega\,\Delta x}\begin{bmatrix} 0 & (c^2 - U^2) \\ 1 & 2U \end{bmatrix} \qquad [B.38]$$

or,

$$[W] \equiv \begin{bmatrix} w_{aa} & w_{aq} \\ w_{qa} & w_{qq} \end{bmatrix} \equiv \begin{bmatrix} 0 & \dfrac{2\,\theta\,\Delta t}{\omega\,\Delta x}(c^2 - U^2) \\ \dfrac{2\,\theta\,\Delta t}{\omega\,\Delta x} & \dfrac{4\,U\,\theta\,\Delta t}{\omega\,\Delta x} \end{bmatrix} \qquad [B.39]$$

Therefore, this method was implemented as a variation of the Petrov-Galerkin approach.

The second line in equation [3.102] represent the extra terms required, some of which are the same as those which arise in the Taylor-Galerkin scheme. Using,

$$\frac{\partial [A]}{\partial x} = \frac{\partial}{\partial x}\begin{bmatrix} 0 & 1 \\ c^2 - U^2 & 2U \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 2\left(gH\dfrac{dH}{dx} - U\dfrac{dU}{dx}\right) & 2\,U \end{bmatrix} \qquad [B.40]$$

these may be combined with those terms from the Taylor-Galerkin scheme to redefine:

$$\begin{bmatrix} b_{aa} & b_{aq} \\ b_{qa} & b_{qq} \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ -\left(gS_o + \dfrac{gH}{B}\dfrac{dB}{dx}\right) - \left(\dfrac{2fQ}{A}\right) + 2\left(gH\dfrac{dH}{dx} - U\dfrac{dU}{dx}\right) & 2\,(f + U) \end{bmatrix} \qquad [B.41]$$

This leads to a variation in not only the Petrov-Galerkin stiffness matrices for the momentum equation, but the mass matrices as well. These become:

$$[S_{qa}] \equiv \left[\int_e \left(w_{qa}\,\omega\,\frac{\Delta x}{2}\,\frac{df_i}{dx}\,f_j \;+\; \theta\,\Delta t\,b_{qa}\,(f_i\,f_j)\right)dx\right] \qquad [B.42]$$

$$[S_{qq}] \equiv \left[\int_e \left\{(f_i\,f_j) + \left(w_{qq}\,\omega\,\frac{\Delta x}{2}\,\frac{df_i}{dx}\,f_j\right) + \theta\,\Delta t\,b_{qq}\,(f_i\,f_j)\right\}dx\right] \qquad [B.43]$$

$$[\mathbf{K}_{qa}] \equiv \left[\int_e \left(\begin{array}{c} \left(-\frac{gH}{2}\frac{d\mathbf{f}_i}{dx}\,\mathbf{f}_j - \left(gS_o + \frac{gH}{2B}\frac{dB}{dx} \right) \mathbf{f}_i\,\mathbf{f}_j \right) \\ + \omega\,\frac{\Delta x}{2}\;w_{qq} \left(-\left(gS_o + \frac{gH}{B}\frac{dB}{dx} \right) \frac{d\mathbf{f}_i}{dx}\,\mathbf{f}_j + a_{qa}\,\frac{d\mathbf{f}_i}{dx}\,\frac{d\mathbf{f}_j}{dx} \right) \\ + \theta\,\Delta t\; b_{qq} \left(-\left(gS_o + \frac{gH}{B}\frac{dB}{dx} \right) \mathbf{f}_i\,\mathbf{f}_j + a_{qa}\,\mathbf{f}_i\,\frac{d\mathbf{f}_j}{dx} \right) \end{array} \right) dx \right] \tag{B.44}$$

and,

$$[\mathbf{K}_{qq}] \equiv \left[\int_e \left(\begin{array}{c} \left(-U\,\frac{d\mathbf{f}_i}{dx}\,\mathbf{f}_j + f\,\mathbf{f}_i\mathbf{f}_j \right) \\ + \;\omega\,\frac{\Delta x}{2}\; \left(w_{qq}\,f\,\frac{d\mathbf{f}_i}{dx}\,\mathbf{f}_j \;+\; (w_{qa} + w_{qq}a_{qq})\,\frac{d\mathbf{f}_i}{dx}\,\frac{d\mathbf{f}_j}{dx} \right) \\ + \theta\,\Delta t\; \left(b_{qq}\,f\,\mathbf{f}_i\mathbf{f}_j \;+\; (b_{qa} + b_{qq}a_{qq})\,\mathbf{f}_i\,\frac{d\mathbf{f}_j}{dx} \right) \end{array} \right) dx \right] \tag{B.45}$$

B.2 Jacobian Terms Used in the Newton-Raphson Iteration

The Jacobian used in the Newton-Raphson iteration scheme has been defined as:

$$[\mathbf{J}] \equiv \left[\frac{\partial\{\mathbf{R}\}}{\partial\,\{\Phi\}} \right] \tag{3.28}$$

If we define

$$[\mathbf{K}'] \equiv \left[\,[\mathbf{S}] + \theta\,\Delta t[\mathbf{K}]\,\right] \tag{B.46}$$

then equation [3.28] may be expanded to:

$$[\mathbf{J}] \equiv [\mathbf{K}'] \left[\frac{\partial\{\Phi\}}{\partial\,\{\Phi\}} \right] + \left[\frac{\partial[\mathbf{K}']}{\partial\,\{\Phi\}} \right] \{\Phi\} = [\mathbf{K}'] + \left[\frac{\partial[\mathbf{K}']}{\partial\,\{\Phi\}} \right] \{\Phi\} \tag{B.47}$$

As discussed in Chapter 3, the first term is often taken to approximate the Jacobian because it requires no additional calculations. Expanding to include the extra terms requires the evaluation of the last term [J']:

$$[\mathbf{J}] \equiv [\mathbf{K}'] + \left[\frac{\partial[\mathbf{K}']}{\partial\{\Phi\}}\right]\{\Phi\} = [\mathbf{K}'] + [\mathbf{J}'] \qquad [B.48]$$

which depends on the method used. Considering the simplest case, the Bubnov-Galerkin, and defining each term individually,

$$[\mathbf{J}'] \equiv \begin{bmatrix} \mathbf{J}_{aa} & \mathbf{J}_{aq} \\ \mathbf{J}_{qa} & \mathbf{J}_{qq} \end{bmatrix} \qquad [B.49]$$

where these represent the terms extra to the [**K'**] contribution such that:

$$[\mathbf{J}_{aa}] \equiv [0] \qquad [B.50]$$

$$[\mathbf{J}_{aq}] \equiv [0] \qquad [B.51]$$

$$[\mathbf{J}_{qa}] \equiv \left[\int_e \theta\,\Delta t \left(-\left(\frac{gH}{2} - U^2\right)\frac{d\mathbf{f}_i}{dx}\,\mathbf{f}_j - \left(\frac{gH}{2B}\frac{dB}{dx} - 2\,U\,f\right)\mathbf{f}_i\mathbf{f}_j\right)dx\right] \qquad [B.52]$$

$$[\mathbf{J}_{qq}] \equiv \left[\int_e \theta\,\Delta t \left(-\,U\,\frac{d\mathbf{f}_i}{dx}\,\mathbf{f}_j + f\,\mathbf{f}_i\mathbf{f}_j\right)dx\right] \qquad [B.53]$$

This may be expanded to include further terms for the more comlex Petrov-Galerkin schemes:

$$\left[\frac{\partial[\mathbf{K}']}{\partial\{\Phi\}}\right]\{\Phi\} \equiv [\mathbf{J}'] = [\mathbf{J}']_{\text{Bubnov-Galerkin}} + [\mathbf{J}']_{\text{Petrov-Galerkin}} \qquad [B.54]$$

Neglecting the dependence of the upwinding matrix on the solution, that is assuming:

$$\frac{\partial[\mathbf{W}]}{\partial\{\Phi\}} \approx [0] \qquad [B.55]$$

then we may write,

$$[\mathbf{J}']_{\text{Petrov-Galerkin}} \approx \omega\,[\mathbf{W}]\,\frac{\partial}{\partial x}\left[\frac{\partial[\mathbf{K}']}{\partial\{\Phi\}}\right]\{\Phi\} \qquad [B.56]$$

such that:

$$[\mathbf{J}_{aa}] \equiv \left[\int_e - \theta\,\Delta t\, w_{aq}\left(\left(\frac{gH}{2B}\frac{dB}{dx} + 2\,U\,f - j_a\right)\frac{d\mathbf{f}_i}{dx}\,\mathbf{f}_j\right)dx\right] \qquad [B.57]$$

$$[\mathbf{J}_{aq}] \equiv \left[\int_e \theta\,\Delta t\, w_{aq}\left((f + j_q)\frac{d\mathbf{f}_i}{dx}\,\mathbf{f}_j\right)dx\right] \qquad [B.58]$$

$$[\mathbf{J}_{qa}] \equiv \left[\int_e \theta\,\Delta t \begin{pmatrix} - \left(\frac{gH}{2} - U^2 +\right)\frac{d\mathbf{f}_i}{dx}\,\mathbf{f}_j - \left(\frac{gH}{2B}\frac{dB}{dx} - 2\,U\,f\right)\mathbf{f}_i\mathbf{f}_j \\ - w_{qq}\left(\frac{gH}{2B}\frac{dB}{dx} + 2\,U\,f - j_a\right)\frac{d\mathbf{f}_i}{dx}\,\mathbf{f}_j \end{pmatrix} dx\right] \qquad [B.59]$$

$$[\mathbf{J}_{qq}] \equiv \left[\int_e \theta\,\Delta t\left(- U\frac{d\mathbf{f}_i}{dx}\,\mathbf{f}_j + f\,\mathbf{f}_i\mathbf{f}_j + w_{qq}\,(f + j_q)\frac{d\mathbf{f}_i}{dx}\,\mathbf{f}_j\right)dx\right] \qquad [B.60]$$

where:

$$j_a \equiv \frac{1}{A}\left\{\frac{\partial A}{\partial x}\left(c^2 + 2U^2\right) - 2\,U\frac{\partial Q}{\partial x}\right\} \qquad [B.61]$$

$$j_q \equiv \frac{2}{A}\left\{\frac{\partial Q}{\partial x} - U\frac{\partial A}{\partial x}\right\} \qquad [B.62]$$

Examining the dependence of the upwinding matrix requires evaluation of the left hand side of equation [B.55]. Writing the upwinding matrix in terms of Froude number, they become for the DG scheme:

$$[\mathbf{W}] \equiv \begin{bmatrix} w_{aa} & w_{aq} \\ w_{qa} & w_{qq} \end{bmatrix} = \begin{bmatrix} 0 & \frac{1}{c}\frac{1}{|\mathbf{F} + 1|} \\ c\frac{-(\mathbf{F} + 1)(\mathbf{F} - 1)}{|\mathbf{F} + 1|} & \frac{2\,\mathbf{F}}{|\mathbf{F} + 1|} \end{bmatrix} \qquad [3.94]$$

and for the CDG scheme:

$$[\mathbf{W}] \equiv \begin{bmatrix} w_{aa} & w_{aq} \\ w_{qa} & w_{qq} \end{bmatrix} = \begin{bmatrix} \frac{1}{2}\frac{\left(1 - \mathbf{F}^2\right)}{|\mathbf{F} + 1| - |\mathbf{F} - 1|} & \frac{1}{2c}\left(\frac{(\mathbf{F} + 1)}{|\mathbf{F} + 1|} - \frac{(\mathbf{F} - 1)}{|\mathbf{F} - 1|}\right) \\ \frac{c}{2}\left(1 - \mathbf{F}^2\right)\left(\frac{(\mathbf{F} + 1)}{|\mathbf{F} + 1|} - \frac{(\mathbf{F} - 1)}{|\mathbf{F} - 1|}\right) & \frac{1}{2}\left(\frac{(\mathbf{F} + 1)^2}{|\mathbf{F} + 1|} - \frac{(\mathbf{F} - 1)^2}{|\mathbf{F} - 1|}\right) \end{bmatrix} \qquad [3.95]$$

From these the derivatives may be calculated using:

$$\frac{\partial[\mathbf{W}]}{\partial\{\Phi\}} = \frac{\partial[\mathbf{W}]}{\partial \mathrm{F}}\,\frac{\partial \mathrm{F}}{\partial\{\Phi\}} \qquad [\mathrm{B}.63]$$

where,

$$\frac{\partial \mathrm{F}}{\partial \mathrm{A}} = -\frac{3}{2}\frac{\mathrm{F}}{\mathrm{A}} \quad \text{and,} \quad \frac{\partial \mathrm{F}}{\partial \mathrm{Q}} = \frac{1}{\mathrm{A\,c}} \qquad [\mathrm{B}.64]$$

Figure 3.7 illutratates the value of [W] as a function of Froude number for the DG scheme, while the results for the CDG scheme are illustrated in Figure 3.8. As the latter figure illustrates, the functions are constant for supercritical flows therefore the Petrov-Galerkin Jacobian defined based on equation [B.55] is exact for supercritical flow when the CDG scheme is used.

B.3 The Non-Conservation and Symmetric Forms

Given the non-conservation form of the St. Venant Equations,

$$\frac{\partial \mathrm{A}}{\partial t} + \frac{\partial \mathrm{Q}}{\partial x} = 0 \qquad [2.6]$$

$$\frac{\partial \mathrm{Q}}{\partial t} + 2\mathrm{U}\frac{\partial \mathrm{Q}}{\partial x} + \left(g\mathrm{H} - \mathrm{U}^2\right)\frac{\partial \mathrm{A}}{\partial x} = g\mathrm{A}\left(\mathrm{S_o} + \frac{\mathrm{H}}{\mathrm{B}}\frac{d\mathrm{B}}{dx} - \mathrm{S_f}\right) \qquad [2.49]$$

implementation of the Petrov-Galerkin method leads to the same mass matrices as presented in equations [B.28] to [B.31]. The stiffness matrices become:

$$[\mathbf{K}_{aa}] \equiv \left[\int_e \omega \frac{\Delta x}{2} w_{aq}\left(-\left(g\mathrm{S_o} + \frac{g\mathrm{H}}{\mathrm{B}}\frac{d\mathrm{B}}{dx}\right)\frac{d\mathbf{f}_i}{dx}\mathbf{f}_j + a_{qa}\frac{d\mathbf{f}_i}{dx}\frac{d\mathbf{f}_j}{dx}\right)dx\right] \qquad [\mathrm{B}.32]$$

which is also unchanged, and

$$[\mathbf{K}_{aq}] \equiv \left[\int_e \left(\mathbf{f}_i \frac{d\mathbf{f}_j}{dx} + \omega \frac{\Delta x}{2} \left(w_{aa} \frac{d\mathbf{f}_i}{dx} \frac{d\mathbf{f}_j}{dx} + w_{aq} f \frac{d\mathbf{f}_i}{dx} \mathbf{f}_j + w_{aq} a_{qq} \frac{d\mathbf{f}_i}{dx} \frac{d\mathbf{f}_j}{dx} \right) \right) dx \right] \qquad [B.65]$$

$$[\mathbf{K}_{qa}] \equiv \left[\int_e \begin{pmatrix} \left(a_{qa} \mathbf{f}_i \frac{d\mathbf{f}_j}{dx} - \left(gS_o + \frac{gH}{B} \frac{dB}{dx} \right) \mathbf{f}_i \mathbf{f}_j \right) \\ + \omega \frac{\Delta x}{2} w_{qq} \left(- \left(gS_o + \frac{gH}{B} \frac{dB}{dx} \right) \frac{d\mathbf{f}_i}{dx} \mathbf{f}_j + a_{qa} \frac{d\mathbf{f}_i}{dx} \frac{d\mathbf{f}_j}{dx} \right) \end{pmatrix} dx \right] \qquad [B.66]$$

$$[\mathbf{K}_{qq}] \equiv \left[\int_e \begin{pmatrix} \left(a_{qq} \mathbf{f}_i \frac{d\mathbf{f}_j}{dx} + f \mathbf{f}_i \mathbf{f}_j \right) \\ + \omega \frac{\Delta x}{2} \left(w_{qq} f \frac{d\mathbf{f}_i}{dx} \mathbf{f}_j + (w_{qa} + w_{qq} a_{qq}) \frac{d\mathbf{f}_i}{dx} \frac{d\mathbf{f}_j}{dx} \right) \end{pmatrix} dx \right] \qquad [B.67]$$

since integration by parts is not used in this case.

Implementation of the CDG method using the symmetric form, is based on,

$$\frac{\partial}{\partial t} \begin{Bmatrix} 2c \\ U \end{Bmatrix} + \begin{bmatrix} U & c \\ c & U \end{bmatrix} \frac{\partial}{\partial x} \begin{Bmatrix} 2c \\ U \end{Bmatrix} + \begin{Bmatrix} 0 \\ g(S_f - S_o) \end{Bmatrix} = \{0\} \qquad [2.60]$$

in which the dependant variables have changed. Neglecting the source terms, these may be written,

$$\frac{\partial}{\partial t} \begin{Bmatrix} 2c \\ U \end{Bmatrix} + \begin{bmatrix} U & c \\ c & U \end{bmatrix} \frac{\partial}{\partial x} \begin{Bmatrix} 2c \\ U \end{Bmatrix} = \{0\} \qquad [B.68]$$

In this case, the convection matrix is redefined and in addition the CDG upwinding matrix is simplified to

$$[\mathbf{W}] = \begin{bmatrix} w_{aa} & w_{aq} \\ w_{qa} & w_{qq} \end{bmatrix} = \frac{1}{2} \begin{bmatrix} \left(\frac{U+c}{|U+c|} + \frac{U-c}{|U-c|} \right) & \left(\frac{U+c}{|U+c|} - \frac{U-c}{|U-c|} \right) \\ \left(\frac{U+c}{|U+c|} - \frac{U-c}{|U-c|} \right) & \left(\frac{U+c}{|U+c|} + \frac{U-c}{|U-c|} \right) \end{bmatrix} \qquad [3.89]$$

The mass matrices are the same as presented in equations [B.28] to [B.31], but the stiffness matrices become:

$$[\mathbf{K}_{aa}] \equiv \left[\int_e \left(U\ f_i \frac{df_j}{dx} + \omega \frac{\Delta x}{2} \left(U\ w_{aa} \frac{df_i}{dx}\frac{df_j}{dx} + c\ w_{aq} \frac{df_i}{dx}\frac{df_j}{dx} \right) \right) dx \right] \qquad [B.69]$$

$$[\mathbf{K}_{aq}] \equiv \left[\int_e \left(c\ f_i \frac{df_j}{dx} + \omega \frac{\Delta x}{2} \left(c\ w_{aa} \frac{df_i}{dx}\frac{df_j}{dx} + U\ w_{aq} \frac{df_i}{dx}\frac{df_j}{dx} \right) \right) dx \right] \qquad [B.70]$$

$$[\mathbf{K}_{qa}] \equiv \left[\int_e \left(c\ f_i \frac{df_j}{dx} + \omega \frac{\Delta x}{2} \left(U\ w_{qa} \frac{df_i}{dx}\frac{df_j}{dx} + c\ w_{qq} \frac{df_i}{dx}\frac{df_j}{dx} \right) \right) dx \right] \qquad [B.71]$$

$$[\mathbf{K}_{qq}] \equiv \left[\int_e \left(U\ f_i \frac{df_j}{dx} + \omega \frac{\Delta x}{2} \left(c\ w_{qa} \frac{df_i}{dx}\frac{df_j}{dx} + U\ w_{qq} \frac{df_i}{dx}\frac{df_j}{dx} \right) \right) dx \right] \qquad [B.72]$$

APPENDIX C

FEM Program Listing

Files included are **Page**

fe.h....................400
femain.c....................402
feio.c....................404
fecore.c....................411
fecom.c....................427
fegauss.c....................439
fetest.c....................441
feinitial.c....................444
femesh.c....................446
festuff.c....................452
feice.c....................455
fewindow.c....................456

fe.h

```
#include "math.h"
#include "stdio.h"
#include "stdlib.h"

#define  MNSF     16
#define  MRNSF     4
#define  MDOF     36
#define  MGPTS     9
#define  NDIMS     2
#define  NPARAMS   3
#define  NBPARAMS  7
#define  NVARS     2
#define  NKEQNS   20

struct shapefuncs {
          int                      dof ;
          double                   f[MNSF] ;
          double                   dfdr[MNSF] ;
          double                   dfds[MNSF] ;
          double                   dfdt[MNSF] ;
          double                   d2fdr[MNSF] ;
          } ;

struct govfuncs {
          int                      dof ;
          double                   detJ ;
          double                   w ;
          double                   p[NPARAMS] ;
          double                   f[MNSF] ;
          double                   dfdx[MNSF] ;
          double                   dfdy[MNSF] ;
          double                   dfdz[MNSF] ;
          double                   d2fdx[MNSF] ;
          } ;

struct gausspts {
          double                   x ;
          double                   y ;
          double                   z ;
          double                   w ;
          } ;

struct    node  {
          int                      n ;
          int                      i ;
          int                      ui ;
          int                      fxc          ;
          double                   x[NDIMS]     ;
          double                   u[NVARS]     ;
          double                   p[NPARAMS]   ;
          double                   uo[NVARS]    ;
          double                   uoo[NVARS]   ;
          float                    f[NDIMS]     ;
          struct node              *nextnp      ;
          } ;

struct    element  {
          int                      n            ;
          int                      vtype        ;
          int                      gtype        ;
          int                      npds         ;
          struct node              *nps[MNSF];
          double                   p[NPARAMS]   ;
          double                   *matrices ;
          float                    elinfo ;
          struct element           *nextelp ;
          } ;

struct    belement  {
          int                      n            ;
          int                      vtype        ;
          int                      gtype        ;
          int                      nnds         ;
          struct node              *nps[MRNSF];
          double                   p[NBPARAMS]           ;
          double                   *matrices ;
```

```
            int                        bcs[NVARS];
            struct belement            *nextbelp ;
            } ;

    struct control {
            int                        trans     ;
            int                        dims      ;
            int                        vars      ;
            int                        params    ;
            int                        bparams ;
            int                        Keqns[NKEQNS];
            int                        nodes     ;
            int                        elms      ;
            int                        belms     ;
            int                        frnds     ;
            int                        ukns      ;
            int                        sym       ;
            int                        nonlin    ;
            } ;

    struct eqnset {
            double                     *Kp ;
            double                     *Ip ;
            double                     *Fp ;
            long                       *diagp ;
            int                        neqns ;
            } ;

    struct pointers {
            struct node                *N ;
            struct element             *El ;
            struct belement            *B ;
            struct node                **iptrs ;
            double                     *S ;
            double                     *M ;
            }  ;

    struct elmpointers {
            double                     *K ;
            double                     *F ;
            double                     *S ;
            double                     *M ;
            double                     *J ;
            int                        n ;
            }  ;

    struct fxnodes {
            int                        i ;
            int                        j[MDOF] ;
            double                     k[MDOF] ;
            double                     f ;
            } ;

    struct transient {
            int                        nsteps ;
            int                        method ;
            int                        iter ;
            double                     dtfac ;
            double                     t ;
            double                     dt ;
            double                     theta ;
            } ;

    struct  RegMesh {
            int                        nx ;
            int                        ny ;
            int                        nbx ;
            int                        nby ;
            int                        elsinbx[MNSF] ;
            int                        elsinby[MNSF] ;
            int                        eltype ;
            int                        maptype ;
            struct control             con ;
            struct blockmap            *firstmap ;
            } ;

    struct blockmap {
            float                      xnodes[MNSF] ;
            float                      ynodes[MNSF] ;
            } ;
```

femain.c

```
#include      "fe.h"

main(argc,argv)
int  argc;
char      **argv;
{
     char       ain ;
     int  nin, errcode, resfilenum, i ;

     initXwindow(argc,argv);
     init_gp() ;
     printf(" Input command (q,i,R,Q,n,l,o,e,a,d,I,S,T,p,u) and number\n") ;
     ain = 't' ;

     while(ain != 'q'){
          printf("> ");
          scanf(" %c %d",&ain,&nin) ;
          switch (ain) {

               case 'i' :
                    errcode = input(nin) ;
                    ScaleXwindow(0);
                    break ;
               case 'R' :
                    ScaleXwindow(nin);
                    break;
               case 'Q' :
                    SetAuxPlot(nin);
                    ScaleXwindow(2);
                    break;
               case 'n' :
                    errcode = test_mesh(nin) ;
                    break ;
               case 'l' :
                    errcode = list_vars(nin) ;
                    break ;
               case 'o' :
                    errcode = output(nin) ;
                    break ;
               case 'e' :
                    errcode = test_elK(nin,-1) ;
                    break ;
               case 'a' :
                    errcode = assemble(nin,0,0,2,0) ;
                    printf(" assemble code = %d\n",errcode);
                    break ;
               case 'd' :
                    errcode = test_trans(nin) ;
                    break ;
               case 'I' :
                    errcode = test_ice(nin) ;
                    Scalexwindow(2) ;
                    break ;
               case 'S' :
                    errcode = test_smooth(nin) ;
                    ScaleXwindow(2) ;
                    break ;
               case 'T' :
                    errcode = test_time(nin) ;
                    break ;
               case 'p' :
                    errcode = set_param(nin) ;
                    break ;
               case 'u' :
                    errcode = set_ptou(nin) ;
                    break ;
          }
     }
     killXwindow();
     exit(0);
}

FILE      *get_fptr(code)
int       code ;

{
    FILE      *fp, *fopen() ;
    char      s[63] ;

    if(code == 1) {
        printf("\n Input Data File Name : ") ;
        scanf(" %s",s) ;
        fp = fopen(s,"r") ;
    }
    else
        fp = stdin ;
    return( fp ) ;
}

FILE      *put_fptr(code)
int       code ;

{
    FILE      *fp, *fopen() ;
    char      s[63] ;

    if(code == 1) {
        printf("\n Input Output File Name : ") ;
        scanf(" %s",s) ;
        fp = fopen(s,"w") ;
    }
    else {
        if(code == 2)
            fp = fopen("fe.out","w") ;
        else
            fp = stdin ;
    }
    return( fp ) ;
}

int       StartMes(astr,bstr,cstr,dstr)
char          *astr, *bstr, *cstr, *dstr ;

{

    printf("%s\n%s\n%s\n%s\n",astr,bstr,cstr,dstr) ;

    return(0);
}

int       UpDateMes(astr,bstr, cstr, dstr)
char          *astr, *bstr, *cstr, *dstr ;

{
    printf("%s\n",dstr) ;
    return(0);
}

int       EndMes()

{
    return(0);
}
```

feio.c

```
#include "fe.h"
#include "ctype.h"

extern struct control  N ;
extern struct pointers gp;
extern struct transient tvals ;

int Nndso, Nelmso, Nbelmso ;

int       input(nin)
int       nin ;

{
    int       err,i ;
    FILE           *fptr, *get_fptr() ;

    if((fptr = get_fptr(nin) ) != NULL) {
        if( (err = get_control(fptr)) == 0) {
            put_control(stdout) ;
            for(i=0;i<N.nodes;i++)
                get_node(fptr,i) ;
            for(i=0;i<N.elms;i++)
                get_elm(fptr,i) ;
            for(i=0;i<N.belms;i++)
                get_belm(fptr,i) ;
            if(fptr != stdin) fclose(fptr) ;
                set_bvalues() ;
        }
        else
            printf(" Control Variable Error %d\n",err) ;
            if(fptr != stdin) fclose(fptr) ;
    }
    else {
        printf(" Unable to open that file\n") ;
        err = -1 ;
        }
    return(err);
}

int       get_control(f)
FILE      *f ;

{
    int        i, j, k, in, ii ;
    long           bigint ;

    j = 0 ;
    for(i=0;i<1;i++) {
        j++ ;
        N.trans = get_int(f) ;
        if(N.trans < 0 || N.trans > 1) break ;
        j++ ;
        if(N.trans == 1)
            get_trans(f) ;
        N.dims = get_int(f) ;
        if(N.dims < 1 || N.dims > NDIMS) break ;
        j++ ;
        N.vars = get_int(f) ;
        if(N.vars < 1 || N.vars > NVARS) break ;
        j++ ;
        for(ii=0;ii<((N.vars+1)*N.vars);ii++)
            N.Keqns[ii] = get_int(f) ;
        j++ ;
        N.params = get_int(f) ;
        if(N.params < 0 || N.params > NPARAMS) break ;
        j++ ;
        N.bparams = get_int(f) ;
        if(N.bparams < 0 || N.bparams > NBPARAMS) break ;
        j++ ;
        if(gp.N != NULL) {
            clear_data() ;
            free(gp.N) ;
            free(gp.iptrs) ;
        }
        N.nodes = get_int(f) ;
        Nndso = N.nodes ;
        bigint = (5 * N.nodes)/4 * sizeof(struct node *) ;
        if(N.nodes < 2 || (gp.iptrs = (struct node **) malloc(bigint))== NULL) break ;
        bigint = (long int) N.nodes * (long int) sizeof(struct node) ;
        if(N.nodes < 2 || (gp.N = (struct node *) malloc(bigint)) == NULL) break ;
        j++ ;
        N.elms = get_int(f) ;
        Nelmso = N.elms ;
        bigint = (long int) N.elms * (long int) sizeof(struct element);
        if(gp.El != NULL)
            free(gp.El) ;
        if(N.elms < 1 || (gp.El = (struct element *) malloc(bigint)) == NULL) break ;
        j++ ;
        N.belms = get_int(f) ;
        Nbelmso = N.belms ;
        bigint = (long int) N.belms * (long int) sizeof(struct belement) ;
        if(gp.B != NULL)
            free(gp.B) ;
        if(N.belms < 1 || (gp.B = (struct belement *) malloc( bigint)) == NULL) break ;

        j = 0 ;
    }
    return(j) ;
}

int     get_trans(f)
FILE *f ;
{
    double get_dbl() ;

    tvals.nsteps = get_int(f) ;
    tvals.dtfac = get_dbl(f) ;
    tvals.t = get_dbl(f) ;
    tvals.dt = get_dbl(f) ;
    tvals.theta = get_dbl(f) ;
    return(0) ;
}

int     clear_data()

{
    int                     i ;
    struct node             *np1, *np2 ;
    struct element          *elp1, *elp2 ;
    struct belement     *belp1, *belp2 ;

    np1 = gp.N ;
    for(i=0;i<N.nodes;i++) {
        np2 = np1->nextnp ;
        if(i >= Nndso)
            free(np1) ;
        np1 = np2 ;
    }
    belp1 = gp.B ;
    for(i=0;i<N.belms;i++) {
        belp2 = belp1->nextbelp ;
        if(i >= Nbelmso)
            free(belp1) ;
        belp1 = belp2 ;
    }
    elp1 = gp.El ;
    for(i=0;i<N.elms;i++) {
        elp2 = elp1->nextelp ;
        if(elp1->matrices != NULL)
            free(elp1->matrices) ;
        if(i >= Nelmso)
            free(elp1) ;
        elp1 = elp2 ;
    }
    return(0) ;
}

int     get_node(f,i)
FILE    *f ;
int     i ;

{
    int                 j ;
    double              get_dbl() ;
    struct node         *nodep ;

    nodep = gp.N + i ;
    nodep->n = get_int(f) ;
    nodep->i = i ;
    nodep->fxc = 0 ;
    gp.iptrs[i] = nodep ;
    for(j=0;j<N.dims;j++)
        nodep->x[j] = get_dbl(f) ;
    for(j=N.dims;j<NDIMS;j++)
        nodep->x[j] = 0.0 ;
    for(j=0;j<N.params;j++)
        nodep->p[j] = get_dbl(f) ;
    for(j=0;j<N.vars;j++)
        nodep->u[j] = get_dbl(f) ;
    nodep->nextnp = nodep + 1 ;
    return(0) ;
}

int         get_elm(f,i)
FILE        *f ;

{
    int                 j, jj, nnds, n, rc ;
    struct element      *elmntp ;
    struct node         *np ;

    elmntp = gp.El + i ;
    rc = 0 ;
    elmntp->n = get_int(f) ;
    elmntp->vtype = get_int(f);
    elmntp->gtype = get_int(f);
    elmntp->nnds =  nsf(elmntp->vtype) ;
    nnds = nsf(elmntp->gtype);
    if(nnds > elmntp->nnds) elmntp->nnds = nnds ;
    if(elmntp->vtype == 229)
        elmntp->nnds = 16 ;
    if(elmntp->vtype == 129)
        elmntp->nnds = 4 ;
    if(elmntp->vtype == 139)
        elmntp->nnds = 6 ;
    for(j=0;j<elmntp->nnds;j++) {
        if((n = get_int(f)) >= 0) {
            np = gp.N ;
            for(jj=0;jj<N.nodes;jj++) {
                if(np->n == n)
                    break ;
                np = np->nextnp ;
            }
            if(np == NULL) {
                printf(" Non-existent node %d referred to in element %d\n",n,elmntp->n) ;
                rc = -1 ;
            }
            elmntp->nps[j] = np ;
        }
        else
            elmntp->nps[j] = NULL ;
    }
    for(j=0;j<N.params;j++)
        elmntp->p[j] = get_dbl(f) ;
    elmntp->matrices = NULL ;
    if(i < N.elms - 1)
        elmntp->nextelp = elmntp + 1 ;
    else
        elmntp->nextelp = NULL ;
    return(rc) ;
}

int         get_belm(f,i)
FILE        *f ;

{
    int                         j, jj, nnds, n, rc ;
    struct belement             *belmntp ;
    struct node                 *np;

    belmntp = gp.B + i ;
    rc = 0 ;
    belmntp->n = get_int(f) ;
    belmntp->vtype = get_int(f);
    belmntp->gtype = get_int(f);
    belmntp->nnds =  nsf(belmntp->vtype) ;
    nnds = nsf(belmntp->gtype);
    if(nnds > belmntp->nnds) belmntp->nnds = nnds ;
    for(j=0;j<belmntp->nnds;j++) {
        n = get_int(f) ;
        np = gp.N ;
        for(jj=0;jj<N.nodes;jj++) {
            if(np->n == n)
                break ;
            np = np->nextnp ;
        }
        if(np == NULL) {
            printf(" Non-existent node %d referred to in element %d\n",n,belmntp->n) ;
            rc = -1 ;
        }
        belmntp->nps[j] = np ;
    }
    for(j=0;j<N.bparams;j++)
        belmntp->p[j] = get_dbl(f) ;
    for(j=0;j<N.vars;j++) {
        belmntp->bcs[j] = get_int(f) ;
    }
    belmntp->matrices = NULL ;
    if(i < N.belms - 1)
        belmntp->nextbelp = belmntp + 1;
    else
        belmntp->nextbelp = NULL ;
    return(rc) ;
}

double  get_dbl(fptr)
FILE    *fptr ;

{
    char        s[25] ;
    int         n ;
    double      atof();

    n = getnumstr(fptr,s) ;
    return( atof(s) ) ;
}

int     get_int(fptr)
FILE    *fptr ;

{
    char        s[25] ;
    int         n ;
    double      atof(), d;

    n = getnumstr(fptr,s) ;
    d = atof(s) ;
    return( (int) d ) ;
}

int  getnumstr(fptr,s)
char     *s ;
FILE     *fptr ;

{
    register int  c ;
    register char *cs ;

    cs = s ;
    while( (c = getc(fptr)) != EOF) {
        if((isdigit(c) != 0) || (c == '.') || (c == '-') || (c == '+')) {*cs++ = c ;
            while( (c = getc(fptr)) != EOF) {
                if((isdigit(c) != 0) || (c == '.') || (c == 'e') || (c == 'E') || (c == '-') || (c == '+') )
                    *cs++ = c ;
                else
                    break ;
            }
            *cs = '\0' ;
            break ;
        }
    }
    return(cs - s) ;
}

int      set_bvalues()

{
    int             i, j ;
    struct node     *nptr ;
    j = 0 ;
    nptr = gp.N ;
    for(i=0;i<N.nodes;i++) {
         nptr->i = i ;
         gp.iptrs[i] = nptr ;
         nptr = nptr->nextnp ;
    }
    N.ukns = N.nodes * N.vars ;
    printf(" Number of unknowns = %d\n",N.ukns) ;
    return(j) ;
}

int       output(nin)
int       nin ;

{
    int                err,i ;
    FILE                    *fptr, *put_fptr() ;
    struct node        *np ;
    struct element     *elp ;
    struct belement *belp ;

    if((fptr = put_fptr(nin) ) != NULL) {
         if( (err = put_control(fptr)) == 0) {
              fprintf(fptr,"\n Node Information \n");
              fprintf(fptr,"\n Node #, Coordinates, Parameters, Variables\n\n");
              np = gp.N ;
              for(i=0;i<N.nodes;i++) {
                   put_node(fptr,i,np) ;
                   np = np->nextnp ;
              }
              fprintf(fptr,"\n Element Information \n");
              fprintf(fptr,"\n Element #, vtype, gtype, nodes\n\n");
              elp = gp.El ;
              for(i=0;i<N.elms;i++) {
                   put_elm(fptr,i,elp) ;
                   elp = elp->nextelp ;
              }
              fprintf(fptr,"\n Boundary Element #, vtype, gtype, nodes, boundary condition codes\n\n");
              belp = gp.B ;
              for(i=0;i<N.belms;i++) {
                   put_belm(fptr,i,belp) ;
                   belp = belp->nextbelp ;
              }
              if(fptr != stdin) fclose(fptr) ;
         }
         else
              printf(" Control Variable Error %d\n",err) ;
              fclose(fptr) ;
    }
    else
         printf(" Unable to open that file\n") ;
    return(err);
}

int       put_control(f)
FILE      *f ;

{
    int i, ii ;

    fprintf(f," Transient analysis = %d\n",N.trans) ;
    if(N.trans == 1){
         fprintf(f," Number of Time Steps = %d\n",tvals.nsteps) ;
         fprintf(f," Delta t Acceleration Factor =%5.3f\n",tvals.dtfac);
         fprintf(f," Time = %5.3f\n",tvals.t) ;
         fprintf(f," Delta t = %5.3f\n", tvals.dt) ;
         fprintf(f," Theta = %5.3f\n", tvals.theta) ;
    }
    fprintf(f," Dimensions = %d\n",N.dims) ;
    fprintf(f," Number of Variables = %d\n",N.vars) ;
    fprintf(f," [K] governing equation numbers \n") ;
         for(i=0;i<N.vars;i++) {
              for(ii=0;ii<N.vars;ii++) {
                   fprintf(f,"\t %d ",N.Keqns[i*(N.vars+1) + ii]) ;
              }
              fprintf(f,"\t\t %d \n",N.Keqns[i*(N.vars+1) + N.vars]) ;
         }
    fprintf(f," Number of Parameters = %d\n",N.params) ;
    fprintf(f," Number of Boundary Parameters = %d\n",N.bparams) ;
    fprintf(f," Number of Nodes = %d\n",N.nodes) ;
    fprintf(f," Number of Elements = %d\n",N.elms) ;
    fprintf(f," Number of  Boundary Elements = %d\n",N.belms) ;
    fprintf(f, " \n") ;

    return(0) ;
}

int     put_node(f,i,nodep)
FILE    *f ;
int     i ;
struct node  *nodep ;

{
    int  j ;

    put_int(f,nodep->n)   ;
    for(j=0;j<N.dims;j++)
         put_dbl(f,(double) nodep->x[j]) ;
    for(j=0;j<N.params;j++)
         put_dbl(f,(double) nodep->p[j]) ;
    for(j=0;j<N.vars;j++)
         put_dbl(f,(double) nodep->u[j]) ;
    fprintf(f," \n");
    return(0) ;
}

int     put_elm(f,i,elmntp)
int     i ;
FILE    *f ;
struct element *elmntp ;

{
    int      j;

    put_int(f,elmntp->n) ;
    put_int(f,elmntp->vtype);
    put_int(f,elmntp->gtype);
    for(j=0;j<elmntp->nnds;j++) {
         if(elmntp->nps[j] != NULL)
              put_int(f,(elmntp->nps[j])->n) ;
         else
              put_int(f,-1) ;
    }
    for(j=0;j<N.params;j++)
         put_dbl(f,(double) elmntp->p[j]) ;
    fprintf(f," \n");
    return(0) ;
}

int     put_belm(f,i,belmntp)
int     i ;
FILE    *f ;
struct belement *belmntp ;

{
    int      j;

    put_int(f,belmntp->n) ;
    put_int(f,belmntp->vtype);
    put_int(f,belmntp->gtype);
    for(j=0;j<belmntp->nnds;j++)
         put_int(f,(belmntp->nps[j])->n) ;
    for(j=0;j<N.bparams;j++)
         put_dbl(f,(double) belmntp->p[j]) ;
    for(j=0;j<N.vars;j++)
         put_int(f,belmntp->bcs[j]) ;
    fprintf(f," \n");
    return(0) ;
}

int     put_dbl(fptr,d)
FILE    *fptr ;
double  d;

{
    fprintf(fptr,"\t%18.8g",d) ;
    return(0) ;
}

int     put_int(fptr,i)
```

```
FILE        *fptr ;
int         i ;

{
     fprintf(fptr,"\t%d",i) ;
     return(0) ;
}
```

fecore.c

```
#include "fe.h"

#define  KE(A,B)  ( *(ep.K + ep.n*(A) + (B)) )
#define  SE(A,B)  ( *(ep.S + ep.n*(A) + (B)) )
#define  ME(A,B)  ( *(ep.M + ep.n*(A) + (B)) )
#define  JE(A,B)  ( *(ep.J + ep.n*(A) + (B)) )
#define  FE(B)     ( *(ep.F + B) )
#define            GRAV     9.81

struct control     N ;
struct pointers    gp;
struct RegMesh     Mesh ;
struct fxnodes     *first_fnp, *next_fnp ;
struct transient   tvals ;
int                            Nukns, Nrfixed,  Fac, *BFnodes, BandWidth ;
struct elmpointers             ep ;
struct eqnset                  eqnsets[4] ;
double                         uchange, *fprime ;

extern double                  UW ;
extern double                  defndp , defelp ;

int  init_gp()

{
    int  i ;

    gp.N = NULL ;
    gp.El = NULL ;
    gp.B = NULL ;
    gp.S = NULL ;
    gp.M = NULL ;
    for(i=0;i<4;i++) {
        eqnsets[i].Kp = NULL ;
        eqnsets[i].Fp = NULL ;
        eqnsets[i].diagp = NULL ;
        eqnsets[i].neqns = 0 ;
    }
    ep.K = NULL ;
    ep.F = NULL ;
    ep.S = NULL ;
    ep.M = NULL ;
    ep.J = NULL ;
    fprime = NULL ;
    first_fnp = NULL ;
    Mesh.nx = 10 ;
    Mesh.ny = 1 ;
    Mesh.nbx = 1 ;
    Mesh.nby = 1 ;
    Mesh.eltype = 111 ;
    Mesh.maptype = 111 ;
    Mesh.firstmap = (struct blockmap *) calloc(24,sizeof(struct blockmap)) ;
    tvals.nsteps = 1 ;
    tvals.t = 0.0 ;
    tvals.dt = 1.0 ;
    tvals.theta = 0.5 ;
    tvals.dtfac = 1.0 ;
    tvals.method = 0 ;
    tvals.iter = 1 ;
    N.sym = 1 ;
    N.nonlin = 1 ;
    localinit() ;
}

int      MkLapMesh(theMesh)
struct   RegMesh  *theMesh ;

{
    N.trans = theMesh->con.trans = 1 ;
    N.dims = theMesh->con.dims = 1 ;
    N.vars = theMesh->con.vars = 2 ;
    N.Keqns[0] = theMesh->con.Keqns[0] = 1 ;
    N.Keqns[1] = theMesh->con.Keqns[1] = 1 ;
    N.params = theMesh->con.params = 3 ;
    N.bparams = theMesh->con.bparams = 7 ;
    defndp = 0.00 ;
    defelp = 0.00 ;
    MakeMesh(theMesh) ;
    return(0) ;
}

/* Method 5 - Taylor Galerkin Method      */

int update_TGKe(gbfp,gtfp,nbf,ntf,np,vcode,theElp )
int                     nbf, ntf, vcode  ;
struct element          *theElp ;
struct node             *np[] ;
struct govfuncs         *gbfp, *gtfp ;

{
    int       i, j, err, n ;

    double    TKaa(), TKaq(), TKqq(), TKqa(), TSaa(), TSqq(), Faa(), Fqq();
    double    Cstar, Q, A, H, U, So, dBdx, ff, B, R, c ;
    double    aaa=0., aaq=0., aqa=0., aqq=0., baa=0., baq=0., bqa=0., bqq=0. ;

    if(vcode<0)
        n = N.vars ;
    else
        n = 1 ;

/*  Compute Parameters Here */

    Q = A = U = So = ff = B = dBdx = 0.0 ;
    for(i=0;i<nbf;i++) {
        A += gbfp->f[i]*np[i]->uo[0] ;
        Q += gbfp->f[i]*np[i]->uo[1] ;
        B += gbfp->f[i]*np[i]->p[2] ;
        ff += gbfp->f[i]*np[i]->p[1] ;
        So += - gbfp->dfdx[i]*np[i]->p[0] ;
        dBdx += gbfp->dfdx[i]*np[i]->p[2] ;
    }

    U = Q/A ;
    H = A/B ;
    R = A/(B+H+H) ;
    if(ff>0.) {
        Cstar = 5.75 * log10(fabs(R/ff)) + 6.2 ;
        ff = fabs(U)/R/Cstar/Cstar ;
    }
    dBdx = H*dBdx/B ;


/*  for linear wave eq'n
    U = 2.93 ;
    U = 0.0 ;
    H = 4.0 ;
    ff = 0.0 ;
*/
    c = fabs(GRAV*H) ;

/*  convection matrix [A] */
    aaa = 0. ;
    aaq = 1. ;
    aqa = c - U*U ;
    aqq = 2. * U ;

/*  derivative of the source terms wrt A and Q */
    baa = 0. ;
    baq = 0. ;
    bqa = -GRAV * (So + dBdx) - (2.* ff * Q)/A ;
    bqq = 2. * ff ;


    for(i=0;i<ntf;i++) {
        for(j=0;j<nbf;j++) {

            SE(i*n,j*n) += TSaa(gtfp, gbfp, i, j) ;
            SE(i*n+1,j*n+1) += TSqq(gtfp, gbfp, i, j) ;

            KE(i*n,j*n) += TKaa(gtfp, gbfp, i, j, So, dBdx, aqa) ;
            KE(i*n,j*n+1) += TKaq(gtfp, gbfp, i, j, ff, aqq) ;
            KE(i*n+1,j*n) += TKqa(gtfp, gbfp, i, j, H, So, dBdx, aqa, aqq, bqq) ;
            KE(i*n+1,j*n+1) += TKqq(gtfp, gbfp, i, j, U, ff, aqq, aqa, aaq, bqa, bqq) ;

        }
        FE(i*n) += Faa(gtfp, gbfp, i) ;
        FE(i*n+1) += Fqq(gtfp, gbfp, i) ;
    }

return(0) ;
}

double      TSaa(v,f,i,j)
int                 i, j ;
struct govfuncs   *v, *f ;

{
    double   t ;
    t = (f->f[i] * f->f[j]) * f->detJ ;
    return(t) ;
}

double      TSqq(v,f,i,j)
int                 i, j ;
struct govfuncs   *v, *f ;

{
    double   t ;
    t = (f->f[i] * f->f[j]) * f->detJ ;
    return(t) ;
}

double      TKaa(v, f, i, j, So, dBdx, aqa)
int                 i, j ;
struct govfuncs   *v, *f ;
double              So, dBdx, aqa ;

{
    double   t ;
    t = tvals.dt / 2. * (-GRAV * (So + dBdx) * f->f[j] + aqa * f->dfdx[j])  * f->dfdx[i] * f->detJ ;
    return(t) ;
}

double      TKaq(v,f,i,j,ff,aqq)
int                 i, j ;
struct govfuncs   *v, *f ;
double              ff, aqq ;

{
    double   t1, t2, t ;
    t1 = - f->dfdx[i] * f->f[j] ;
    t2 = tvals.dt/2. * ( ff * f->f[j] + aqq * f->dfdx[j]) * f->dfdx[i] ;
    t = (t1 + t2) * f->detJ ;
    return(t) ;
}

double      TKqa(v, f, i, j, H, So, dBdx, aqa, aqq, bqq)
int                 i, j ;
struct govfuncs   *v, *f ;
double              H, So, dBdx, aqa, aqq, bqq ;

{
    double   t1, t2,t3, t4, t ;
    t1 = - GRAV * H / 2. * f->dfdx[i] * f->f[j] ;
    /* FOR LINEAR WAVE
    t1 = - GRAV * H * f->dfdx[i] * f->f[j] ;*/
    t2 = - GRAV * (So + dBdx/2.) * f->f[i] * f->f[j] ;
    t3 = tvals.dt/2. * bqq * (- GRAV * (So + dBdx) * f->f[i] * f->f[j] + aqa * f->f[i] );
    t4 = tvals.dt/2. * (aqq * (- GRAV * (So + dBdx) * f->dfdx[i] * f->f[j]) + aqa * aqq * f->dfdx[i] * f-
        >dfdx[j] );
    t = (t1 + t2 + t3 + t4) * f->detJ ;
    return(t) ;
}

double      TKqq(v ,f , i, j, U, ff, aqq, aqa, aaq, bqa, bqq)
int                 i, j ;
struct govfuncs   *v, *f ;
double              U, ff, aqq, aaq, aqa, bqa, bqq ;

{
    double   t1, t2, t3, t ;
    t1 = (- U * f->dfdx[i] * f->f[j] + ff * f->f[i] * f->f[j]) ;
    t2 = tvals.dt / 2. *( bqq * ff * f->f[i] * f->f[j] + (bqa+aqq*bqq) * f->f[i] * f->dfdx[j] ) ;
    t3 = tvals.dt / 2. * (aqq * ff * f->dfdx[i] * f->f[j] + (aqa*aaq + aqq*aqq) * f->dfdx[i] * f->dfdx[j] )
        ;
    t = (t1 + t2 + t3) * f->detJ ;
    return(t) ;
}

/* Implicit FEM schemes    */

int update_PGKe(gbfp,gtfp,nbf,ntf,np,vcode,theElp )
int                     nbf, ntf, vcode  ;
struct element          *theElp ;
struct node             *np[] ;
struct govfuncs    *gbfp, *gtfp ;

{
    int        i, j, err, n ;

    double   Kaa(),Kaq(), Kqq(), Kqa(), Saa(), Saq(),Sqq(), Sqa(), Jqq(), Jqa(), Faa(), Fqq();
    double   Cstar, Qnew, Anew, Hnew, Unew, So, dBdx, ff, B, R, c, dHdx, dUdx ;
    double   Qold, Aold, Hold, Uold ;
    double   waa=0., waq=0., wqa=0., wqq=0., baa=0., baq=0., bqa=0., [illegible]=0. ;

    if(vcode<0)
        n = N.vars ;
    else
        n = 1 ;

    for(i=0;i<nbf;i++)
        gtfp->f[i] = UW * gbfp->dfdx[i] * gbfp->detJ ;

/*  Compute Parameters Here */

    Qold = Aold = 0.0 ;
    Qnew = Anew = Unew = So = ff = B = dBdx = dHdx = dUdx = 0.0 ;
    for(i=0;i<nbf;i++) {
        Aold += gbfp->f[i]*np[i]->uo[0] ;
        Qold += gbfp->f[i]*np[i]->uo[1] ;
        Anew += gbfp->f[i]*np[i]->u[0] ;
        Qnew += gbfp->f[i]*np[i]->u[1] ;
        B += gbfp->f[i]*np[i]->p[2] ;
        ff += gbfp->f[i]*np[i]->p[1] ;
        So += - gbfp->dfdx[i]*np[i]->p[0] ;
        dBdx += gbfp->dfdx[i]*np[i]->p[2] ;
        dHdx += gbfp->dfdx[i]*(np[i]->u[0]/np[i]->p[2]) ;
        dUdx += gbfp->dfdx[i]*(np[i]->u[1]/np[i]->u[0]) ;

    }

    Uold = Qold/Aold ;
    Hold = Aold/B ;
    Unew = Qnew/Anew ;
    Hnew = Anew/B ;
    R = Anew/(B+Hnew+Hnew) ;
    if(ff>0.) {
        Cstar = 5.75 * log10(fabs(R/ff)) + 6.2 ;
        ff = fabs(Unew)/R/Cstar/Cstar ;
    }
    dBdx = Hnew*dBdx/B ;


/*  for linear wave eq'n
    Unew = 2.93 ;
    Unew = 0.0 ;
    Hnew = 4.0 ;
    ff = 0.0 ;
*/
    c = sqrt(fabs(GRAV*Hnew)) ;


    if(tvals.method ==3) {
/*  Method 3 - DG Method  */
        waa = 0. ;
        waq = 1.0/fabs(Unew + c) ;
        wqa = (c*c - Unew*Unew)/fabs(Unew + c);
        wqq = (2.0 * Unew)/fabs(Unew + c) ;
    }

    if(tvals.method ==4) {
/*  Method 4 - CDG Method  */
        waa = (c*c - Unew*Unew) * (1./fabs(Unew+c) - 1./fabs(Unew-c)) /2./c;
        waq = ( (Unew+c)/fabs(Unew+c) - (Unew-c)/fabs(Unew-c) ) /2./c ;
        wqa = ( (c*c - Unew*Unew) * ( (Unew+c)/fabs(Unew+c) - (Unew-c)/fabs(Unew-c) ) )/2./c ;
        wqq = ( (Unew+c)*(Unew+c)/fabs(Unew+c) - (Unew-c)*(Unew-c)/fabs(Unew-c) ) /2./c ;
    }
    if(tvals.method ==6) {
/*  Method VI - Least-Squares Method (UW set to theta dt/|J|)  */
        waa = 0.0 ;
         waq = (c*c - Unew*Unew) * tvals.theta * tvals.dt / gbfp->detJ/UW ;
        wqa = 1.0 * tvals.theta * tvals.dt / gbfp->detJ / UW ;
        wqq = 2.0 * Unew * tvals.theta * tvals.dt / gbfp->detJ / UW ;
        baa = 0. ;
        baq = 0. ;
        bqa = -GRAV * (So + dBdx) - (2.* ff * Qnew)/Anew + GRAV * dHdx - 2. * Unew * dUdx;
        bqq = 2. * (ff+dUdx);
    }

    for(i=0;i<ntf;i++) {
        for(j=0;j<nbf;j++) {

            SE(i*n,j*n) += Saa(gtfp, gbfp, i, j, waa) ;
            SE(i*n,j*n+1) += Saq(gtfp, gbfp, i, j, waq) ;
            SE(i*n+1,j*n+1) += Sqq(gtfp, gbfp, i, j, wqq, bqq) ;
            SE(i*n+1,j*n) += Sqa(gtfp, gbfp, i, j, wqa, bqa) ;

            KE(i*n,j*n) += Kaa(gtfp, gbfp, i, j, Hnew, So, Unew, dBdx, waq);
            KE(i*n,j*n+1) += Kaq(gtfp, gbfp, i, j, Unew, ff, waa, waq) ;
            KE(i*n+1,j*n) += Kqa(gtfp, gbfp, i, j, Hnew, So, Unew, dBdx, wqq, bqq) ;
            KE(i*n+1,j*n+1) += Kqq(gtfp, gbfp, i, j, Unew, ff, wqa, wqq, bqa, bqq) ;


            if(N.nonlin > 0) {
                JE(i*n+1,j*n) += Jqa(gtfp, gbfp, i, j, Hnew, Unew, ff,dBdx);
                JE(i*n+1,j*n+1) += Jqq(gtfp, gbfp, i, j, Unew, ff) ;
            }

        }
/*      source/sink term */
        FE(i*n) += Faa(gtfp, gbfp, i) ;
        FE(i*n+1) += Fqq(gtfp, gbfp, i) ;
    }

return(0) ;
}


double          Saa(v,f,i,j,waa)
int                         i, j ;
struct govfuncs     *v, *f ;
double                      waa ;

{
    double   t ;
    t = ( f->f[i] * f->f[j] + waa * v->f[i] * f->f[j] ) * f->detJ ;
    return(t) ;
}


double          Saq(v,f,i,j,waq)
int                         i, j ;
struct govfuncs     *v, *f ;
double                      waq ;

{
    double   t ;
    t = waq * v->f[i] * f->f[j] * f->detJ ;
    return(t) ;
}


double          Sqq(v,f,i,j,wqq, bqq)
int                         i, j ;
struct govfuncs     *v, *f ;
double                      wqq, bqq ;

{
    double   t ;
    t = (f->f[i] * f->f[j] + (wqq * v->f[i] + bqq*tvals.theta*tvals.dt* f->f[i]) * f->f[j]) * f->detJ ;
    return(t) ;
}


double          Sqa(v,f,i,j,wqa,bqa)
int                         i, j ;
```

```
struct govfuncs    *v, *f ;
double                 wqa, bqa ;

{
    double   t ;
    t = (wqa*v->f[i] + bqa*tvals.theta*tvals.dt*f->f[i]) * f->f[j] *    f->detJ ;
    return(t) ;
}


double        Kaa(v,f,i,j,H,So,U,dBdx,waq)
int                    i, j ;
struct govfuncs    *v, *f ;
double                 H, So, U, dBdx, waq ;

{
    double   t1, t2, t ;
    t1 = (GRAV * H -  U * U) * waq * v->f[i] * f->dfdx[j] ;
    t2 = -GRAV * (So + dBdx) * waq * v->f[i] * f->f[j] ;
    t = (t1 + t2)  * f->detJ ;
    return(t) ;
}


double        Kaq(v,f,i,j,U,ff,waa,waq)
int                    i, j ;
struct govfuncs    *v, *f ;
double                 U, ff, waa, waq ;

{
    double   t1, t2, t3, t4, t ;
    t1 = - f->dfdx[i] * f->f[j] ;
    t2 = waa * v->f[i] * f->dfdx[j] ;
    t3 = waq * ff * v->f[i] * f->f[j] ;
    t4 = waq * 2.0 * U * v->f[i] * f->dfdx[j] ;
    t = (t1 + t2 + t3 + t4) * f->detJ ;
    return(t) ;
}


double        Kqa(v,f,i,j,H,So,U,dBdx,wqq,bqq)
int                    i, j ;
struct govfuncs    *v, *f ;
double                 H,So,U,dBdx,wqq, bqq ;
{
    double   t1, t2, t3, t4, t5, t ;
    t1 = - GRAV * H / 2. * f->dfdx[i] * f->f[j] ;
    /* FOR LINEAR WAVE
    t1 = - GRAV * H * f->dfdx[i] * f->f[j] ;*/
    t2 = - GRAV * (So + dBdx/2.) * f->f[i] * f->f[j] ;
    t3 = (GRAV * H -  U * U) * wqq * v->f[i] * f->dfdx[j] ;
    t4 = -GRAV * (So + dBdx) * wqq * v->f[i] * f->f[j] ;
    t5 = tvals.dt * tvals.theta * bqq * (- GRAV * (So + dBdx) * f->f[i] * f->f[j] + (GRAV * H -  U * U) *
        f->f[i] * f->dfdx[j] ) ;
    t = (t1 + t2 + t3 + t4 + t5) * f->detJ ;
    return(t) ;
}


double        Kqq(v,f,i,j,U,ff,wqa,wqq,bqa,bqq)
int                    i, j ;
struct govfuncs    *v, *f ;
double                 U, ff, wqa, wqq, bqa, bqq ;

{
    double   t1, t2, t3, t4, t5, t ;
    t1 = (- U * f->dfdx[i] * f->f[j] + ff * f->f[i] * f->f[j]) ;
    t2 = wqa * v->f[i] * f->dfdx[j] ;
    t3 = wqq * 2.0 * U * v->f[i] * f->dfdx[j] ;
    t4 = wqq * ff * v->f[i] * f->f[j] ;
    t5 = tvals.dt * tvals.theta *( bqq * ff * f->f[i] * f->f[j] + (bqa+2.*U*bqq) * f->f[i] * f->dfdx[j] ) ;
    t = (t1 + t2 + t3 + t4 + t5) * f->detJ ;
    return(t) ;
}


double        Jqq(v,f,i,j,U,ff)
int                    i, j ;
struct govfuncs    *v, *f ;
double                 U, ff ;
```

```
{
    double   t1, t2, t ;
    t1 = -U * f->dfdx[i] * f->f[j] ;
    t2 = ff * f->f[i] * f->f[j] ;
    t = (t1 + t2) * f->detJ ;
    return(t) ;
}


double        Jqa(v,f,i,j,H,U,ff,dBdx)
int                        i, j ;
struct govfuncs    *v, *f ;
double                     H, U ; ff, dBdx ;
{
    double   t1, t2, t3, t ;
    t1 = -(GRAV * H / 2.0 - U * U) * f->dfdx[i] * f->f[j] ;
    t2 = - 2.0 * U * ff * f->f[i] * f->f[j] ;
    t3 = - GRAV * dBdx / 2. * f->f[i] * f->f[j] ;
    t = (t1 + t2 + t3) * f->detJ ;
    return(t) ;
}


double        Faa(v,f,i)
int                        i ;
struct govfuncs    *v, *f ;

{
    double   t ;
    t = 0.;
    return(t) ;
}

double        Fqq(v,f,i)
int                        i ;
struct govfuncs    *v, *f ;

{
    double   t ;
    t = 0.;
    return(t) ;
}


int  update_BKe(elmntp,gbf,gtf,nbf,ntf,np,nvar)
int                         nbf, ntf ;
struct node                 *np[] ;
struct belement     *elmntp ;
struct shapefuncs        *gbf, *gtf ;

{
    int      i, j, n ;
    double   Qold, Aold, Hold, Uold, B, phi ;

    if(nvar<0)
        n = N.vars ;
    else
        n = 1 ;

    Qold = 0.0 ;
    Hold = 0.0 ;
    Aold = 0.0 ;
    phi = 0.0 ;
    B = 0.0 ;
    for(i=0;i<nbf;i++) {
        Aold += gbf->f[i]*np[i]->u[0] ;
        Qold += gbf->f[i]*np[i]->u[1] ;
        B += gbf->f[i]*np[i]->p[2] ;
    }

    Uold = Qold/Aold ;
    phi = (GRAV * Aold / B / 2.0) ;
    Hold = Aold / B  ;

    for(i=0;i<ntf;i++) {
        for(j=0;j<nbf;j++) {

            if (tvals.theta<0.005) {
                    SE(i*n,j*n) += elmntp->p[6] * (1.E18 * elmntp->p[0]) ;
                    SE(i*n,j*n+1) += elmntp->p[6] * (1.E18 * elmntp->p[1]) ;
                    SE(i*n+1,j*n) += elmntp->p[6] * (1.E18 * elmntp->p[3]) ;
        /*          FOR LINEAR WAVE   SE(i*n+1,j*n) += elmntp->p[6] * (1.E18 * elmntp->p[3]) ;*/
                    SE(i*n+1,j*n+1) += elmntp->p[6] * (1.E18 * elmntp->p[4]);
            }
            else {
                    KE(i*n,j*n) += elmntp->p[6] * (0.0 + 1.E18 * elmntp->p[0]) ;
                    KE(i*n,j*n+1) += elmntp->p[6] * (1.0 + 1.E18 * elmntp->p[1]) ;
                    KE(i*n+1,j*n) += elmntp->p[6] * (phi + 1.E18 * elmntp->p[3]) ;
            /*       FOR LINEAR WAVE  KE(i*n+1,j*n) += elmntp->p[6] * (1.E18 * elmntp->p[3]) ;*/
                    KE(i*n+1,j*n+1) += elmntp->p[6] * (Uold * 1.E18 * elmntp->p[4]) ;
            }

            if(N.nonlin > 0) {
                JE(i*n+1,j*n) += 0.0 ;
                JE(i*n+1,j*n+1) += 0.0 ;
            }

        }
            if (tvals.theta>0.005) {

                FE(i*n) += elmntp->p[6] * (Aold * 1.E18 * elmntp->p[2]) ;
                FE(i*n+1) += elmntp->p[6] * (Qold * 1.E18 * elmntp->p[5]) ;
            }

    }
}


int  transient(nvar,elcode,tol)
int        nvar,elcode ;
double     tol ;

{
    int  i, j, jj, itnum, day, hr, min, sec, hnum1, hnum2 ;
    long            time ;
    struct   node    *nodep, *maxnodep ;
    FILE            *fptr, *put_fptr() ;


    N.trans = 1 ;

    set_ptou(0) ;

    if((fptr = put_fptr(1) ) != NULL){

        if(tvals.method ==0 || tvals.method > 6)
            printf("IBP F#6 - Bubnov-Galerkin\n") ;
        if(tvals.method ==3)
            printf("IBP F#6 - Method 3 - Dissipative Galerkin\n") ;
        if(tvals.method ==4)
            printf("IBP F#6 - Method 4 - Char.-Dissipative-Galerkin\n") ;
        if(tvals.method ==5)
            printf("IBP F#6 - Method 5 - Taylor Galerkin\n") ;
        if(tvals.method ==6)
            printf("IBP F#6 - Method 6 - Least-Squares\n") ;

        fprintf(fptr,"Time Step =%g\tTheta = %g\n",tvals.dt,tvals.theta) ;
        fprintf(fptr,"\n") ;

        for(i=0;i<tvals.nsteps;i++) {
            tvals.t += tvals.dt ;
            uchange = 1.0 ;
            itnum = 0 ;
            for(j=0;j<15;j++) {
                if(uchange<tol)
                    break ;
                assemble (nvar,0,0,elcode,itnum) ;

                solve(nvar,2,0) ;
                itnum += 1 ;
            }

            if ((i+1)%tvals.iter == 0) {
               DrawElev();
               printf("Time step=%d,\tt=%fs\tIterations=%d\tuchange=%g\n" ,i+1,tvals.t,itnum,uchange) ;
               fprintf(fptr,"Time step=%d,\tt=%fs\tIterations=%d\tuchange%g\n",i+1,tvals.t,itnum,uchange);
               for(jj=0;jj<N.nodes;jj++)
                  fprintf(fptr,"%g\t%g\t%g\n",gp.iptrs[jj]->x[0] ,gp.iptrs[jj]->u[0],gp.iptrs[jj]->u[1]);
            }

            tvals.dt = tvals.dt * tvals.dtfac ;
            set_ptou(0) ;
    }

    fclose(fptr) ;

    return(0) ;
}

int     assemble(varnum,code,nset,elcode,itnum)
int     varnum, code, nset, elcode, itnum  ;

{
    double                  *K_init(), *L_init(), *F_init() ;
    struct fxnodes          *fn_init() ;
    struct element          *elp, tel ;
    struct belement         *belp, tbel ;
    double                  *p ;
    int                     i,j,ns,ntf,istart,iend,ind ;
    long                    ij ;
    char                    m1[255], m2[255], m3[255], m4[255] ;
    FILE                    *fptr, *put_fptr() ;

    istart = 0 ;
    iend = N.elms ;
    if(N.traps == 0)
        tvals.dt = 1.0 ;

    if(code< 2) {

        if(eqnsets[nset].Kp != NULL)
            free(eqnsets[nset].Kp) ;

        if( (eqnsets[nset].Kp = K_init(varnum,&eqnsets[nset].diagp,code)) == NULL )
            return(-1) ;

    if(N.sym != 0) {
            if(eqnsets[nset].Lp != NULL )
                free(eqnsets[nset].Lp) ;
            if( (eqnsets[nset].Lp = L_init(varnum,& eqnsets[nset].diagp, code)) == NULL )
                return(-1) ;
        }

        eqnsets[nset].neqns = Nukns ;
/*      sprintf(m1," Assembling K matrix and F vector ");
        sprintf(m2," Number of equations to be solved = %d ",Nukns) ;
        sprintf(m3," Elements in K matrix = %ld ",*(eqnsets[nset].diagp+ eqnsets[nset].neqns-1)+1) ;
        sprintf(m4," ") ;
        StartMes(m1,m2,m3,m4) ;
*/
        if(eqnsets[nset].Fp != NULL) {
            free(eqnsets[nset].Fp) ;
        }
        if( ( eqnsets[nset].Fp = F_init(Nukns) ) == NULL  )
            return(-1) ;
    }
    else if (code == 2) {
        ind = 0 ;
        for(i=0;i<N.nodes;i++) {
            gp.iptrs[i]->ul = ind ;
            ind++ ;
        }
    }

    if(code < 1) {
/*
        sprintf(m4," Assembling Element # 0 of %d",N.elms) ;
        UpDateMes(m1,m2,m3,m4) ;
*/

        if( (ep.n = ep_init() ) == 0 )
            return(-1) ;
    }
    for(i=0;i<eqnsets[nset].neqns;i++)
        *(eqnsets[nset].Fp + i) = 0.0 ;
    elp = gp.El ;
    for(i=istart;i<iend;i++) {

        if( i % 10 ==  0 && code < 2 && i != 0) {
/*              sprintf(m4," Assembling Element # %d of %d ",i,N.elms) ;
```

```
                UpDateMes(m1,m2,m3,m4) ;
*/          }

            ns = get_PGKe(elp,&tel,varnum,elcode,&ntf,itnum) ;
            put_Ke(ns,ntf,&tel,varnum,eqnsets[nset].Kp,eqnsets[nset].Lp,
                eqnsets[nset].Fp,eqnsets[nset].diagp,code) ;
            elp = elp->nextelp ;
        }

        belp = gp.B ;
        for(i=0;i<N.belms;i++) {
            ns = get_bKe(belp,&tbel,varnum,elcode,&ntf,itnum) ;
            if(ns > 0 )
                put_Ke(ns,ntf,belp,varnum,eqnsets[nset].Kp,eqnsets[nset].Lp,
                    eqnsets[nset].Fp,eqnsets[nset].diagp,code) ;
            belp = belp->nextbelp ;
        }

        if(code < 2){

/*          printf("# of entries in skyline matrix = %ld\n",*(eqnsets[nset].diagp + eqnsets[nset].neqns-1)+1 );
            if((fptr = put_fptr(1) ) != NULL){
                for (lj = 0; lj < *(eqnsets[nset].diagp + Nukns-1)+1 ;lj++)
                    fprintf(fptr,"\t%ld\t\t%g\t\t%g\n",lj, *(eqnsets[nset].Kp+lj),*(eqnsets[nset].Lp + lj)) ;
                for (j = 0; j < Nukns ; j++)
                    fprintf(fptr,"\n\t%d\t\t%g\n",j,*(eqnsets[nset].Fp + j)) ;
            }
*/
            EndMes() ;
        }

        return(0) ;
}

double    *K_init(nvar,thediagp,code)
int       nvar, code ;
long          *(*thediagp) ;

{
    int             nj, i, j, nf , ind, n;
    double          *kp, *dp ;
    unsigned        elsize ;
    long            *bigint, nelem,*diagp, il ;

    if(nvar<0)
        n = N.vars ;
    else
        n = 1 ;
    nelem = N.nodes * n ;
    Nukns = nelem ;
    if(*thediagp != NULL)
        free(*thediagp) ;
    elsize = sizeof(long) ;
    if((*thediagp = (long int *) calloc((int)nelem,sizeof(long)))==NULL)
        return(NULL) ;
    diagp = *thediagp ;
    *diagp = 0 ;
    ind = 0 ;
    gp.iptrs[0]->ui = ind ;
    diagp++ ;
    ind++ ;
    for(i=1;i<Nukns;i++) {
        gp.iptrs[i/n]->ui = ind ;
        ind++ ;
        if(code == 0) {
/*
            nf =  (mpd_i((i/n),nvar)  + 1) * n ;
*/
            if(ind < BandWidth)
                nf = ind ;
            else
                nf = BandWidth ;
        }
        else if(code == -1)     /*for assembling a diagonal matrix only*/
            nf = 1 ;
        else if(code == 2) {/*for assembling a pentadiagonal matrix*/
            if(ind == 1)
                nf = 1;
            else if(ind == 2)
                nf = 2  ;
            else
```

```
                nf = 3 ;
            }
            else
                nf = 2 * n;
            *diagp = *(diagp-1) + nf ;
            diagp++ ;
        }
        nelem = *(diagp-1) + 1 ;
        elsize = sizeof(double) ;
        bigint =  nelem * (long) elsize  ;
        kp =  dp = (double *) malloc(bigint)  ;
/*
        printf(" memory for U matrix = %ld\n",bigint);
*/
        for(il=0;il<nelem;il++)
            *(dp++) = 0.0 ;
        return(kp) ;
}

double   *L_init(nvar,thediagp,code)
int         nvar,code ;
long            *(*thediagp) ;

{
        int             nj, i, j, nf , ind;
        double          *kp, *dp ;
        unsigned        elsize ;
        long                    bigint, nelem, il, *diagp ;

        diagp = *thediagp ;
        for(i=0;i<Nukns;i++)
                diagp++ ;

        nelem = *(diagp-1) + 1 ;
        elsize = sizeof(double) ;
        bigint =  nelem * (long) elsize  ;
        kp =  dp = (double *) malloc(bigint)  ;
/*
        printf(" memory for L matrix = %ld\n",bigint);
*/
        for(il=0;il<nelem;il++)
            *(dp +) = 0.0 ;
        return(kp) ;
}

double   *F_init(num)
int        num ;

{
        double   *p ;
        unsigned        nelem, elsize ;

        nelem = num ;
        elsize = sizeof(double) ;
        p = (double *) calloc(nelem,elsize) ;
        return(p) ;
}

struct fxnodes    *fn_init(nvar)
int        nvar ;

{
        int        i, j ;
        struct fxnodes     *p, *ap ;
        unsigned        nelem, elsize ;

        Nrfixed = 0 ;
        for(i=0;i<N.nodes;i++) {
            if(isvarfixed(i,nvar) == -1)
                Nrfixed++ ;
        }
        nelem = Nrfixed ;
        elsize = sizeof(struct fxnodes) ;
        ap = p = (struct fxnodes *) calloc(nelem,elsize) ;
        for(i=0;i<nelem;i++) {
            ap->i = -1 ;
            for(j=0;j<MDOF;j++)
                ap->j[j] = -1 ;
            ap++ ;
        }
        return(p) ;
```

```
}

int        ep_init()

{
     int                    i, ns, nsm ;
     struct element         *elp ;

     if(ep.K != NULL)
          free(ep.K) ;
     elp = gp.El ;
     nsm = 0 ;

     for(i=0;i<N.elms;i++) {
          nsm = ( (ns = nsf(elp->vtype)) > nsm ) ? ns : nsm ;
          elp = elp->nextelp ;
     }
     nsm *= N.vars ;
/*
     printf("Size of Element Matrices = %d\n",nsm) ;
*/
     if( (ep.K = (double *) calloc(nsm*nsm, sizeof(double)) ) == NULL )
          return(0) ;
     if(ep.F != NULL)
          free(ep.F) ;
     if( (ep.F = (double *) calloc(nsm, sizeof(double)) ) == NULL )
          return(0) ;
     if(N.trans > 0) {
          if(ep.S != NULL)
               free(ep.S) ;
          if( (ep.S = (double *) calloc(nsm*nsm, sizeof(double)) ) == NULL )
               return(0) ;
     }
     if(N.trans > 1) {
          if(ep.M != NULL)
               free(ep.M) ;
          if( (ep.M = (double *) calloc(nsm*nsm, sizeof(double)) ) == NULL )
          return(0) ;
     }
     if(N.nonlin > 0) {
          if(ep.J != NULL)
               free(ep.J) ;
          if( (ep.J = (double *) calloc(nsm*nsm, sizeof(double)) ) == NULL )
          return(0) ;
     }
     return(nsm) ;
}

int        put_Ke(ns,nt,elmntp,nvar,Kp,Lp,Fp,diagp,code)
int                  ns, code ;
double               *Kp,  *Lp, *Fp ;
struct element *elmntp ;
long                     *diagp ;

{
     int  i, in, nvars, j, ii, jj, jn, frow, fcol, row, col, tj, isv ;
     struct node          *inp, *jnp ;
     struct fxnodes *the_fnp ;

     if(nvar < 0)
          nvars = N.vars ;
     else
          nvars = 1 ;

     for(i=0;i<nt;i++) {
          inp = elmntp->nps[i] ;
          frow = inp->i * nvars ;
          for(j=0;j<ns;j++) {
               jnp = elmntp->nps[j] ;
               fcol = jnp->i * nvars;
               for(ii=0;ii<nvars;ii++) {
                    row = frow + ii ;
                    for(jj=0;jj<nvars;jj++) {
                         col = fcol + jj ;
                         if( (col >= row) && ( code < 2 ) && ( code > -1 ) ) {
                              if(N.trans == 0)
                                   *(Kp + *(diagp+col) - col + row ) +=
                                        KE(i*nvars+ii,j*nvars+jj) ;
                              else
                                   *(Kp + *(diagp+col) - col + row ) +=
                                        SE(i*nvars+ii,j*nvars+jj) ;
```

```
                        }

                        if( (col <= row) && (N.sym != 0 ) ) {
                            if(N.trans == 0)
                                *(Lp + *(diagp+row) - row + col ) += KE(i*nvars+ii,j*nvars+jj) ;
                            else
                                *(Lp + *(diagp+row) - row + col ) += SE(i*nvars+ii,j*nvars+jj) ;
                        }

                        if( (col == row) && ( code == -1 ) ) {
                            if(N.trans == 0)
                                *(Kp + row ) += KE(i*nvars+ii,j*nvars+jj) ;
                            else
                                *(Kp + row ) += SE(i*nvars+ii,j*nvars+jj) ;
                        }
                        if(  (code == 2 ) || code == -1 ) {
                            *(Fp + row ) += -KE(i*nvars+ii,j*nvars+jj)*jnp->u[jj];
                        }
                    }
                }
            }
            for(ii=0;ii<nvars;ii++) {
                row = frow + ii ;
                *(Fp + row) += FE(i*nvars+ii) ;
            }
        }
}

int     put_BFKe(ns,nt,elmntp,nvar,Kp,Fp,diagp,code)
int     ns, code ;
double  *Kp, *Fp ;
struct element *elmntp ;
long    *diagp ;

{
    int i, in,iv, j, jn, jv,  row, col, tj, isv ;
    struct node  *inp, *jnp ;
    struct fxnodes *the_fnp ;

    for(i=0;i<nt;i++) {
        row = code + i ;
        for(j=0;j<ns;j++) {
            col = code + j ;
            if( col >= row) {
                *(Kp + *(diagp+col) - col + row ) += KE(i,j) ;
            }
        }
    }
}

int     get_PGKe(elmntp,theElp,nvar,code,ntf,itnum)
struct element          *elmntp, *theElp ;
int                     nvar, code, *ntf, itnum ;

{
    struct gausspts   g[MGPTS] ;
    struct shapefuncs     bf, tf, *fgp;
    struct node           *np[MNSF] ;
    struct govfuncs   gbf, gtf ;
    int                   i, j, nbf, ni, nm, dir, n, jj ;
    double                tdt, mtdt, sdiag ;
    double                *fptr ;

    if(nvar<0)
        n = N.vars ;
    else
        n = 1 ;
    initm(ep.K,ep.n*ep.n) ;
    if(N.trans > 0)
        initm(ep.S,ep.n*ep.n) ;
    if(N.nonlin > 0)
        initm(ep.J,ep.n*ep.n) ;
    initm(ep.F,ep.n) ;

    ElPick(elmntp,theElp,&ntf,&dir) ;
    nbf = theElp->nnds ;

    if(code > 0 || elmntp->matrices == NULL || N.nonlin > 0) {
        ni = get_gspts(theElp->vtype,g) ;
        for(j=0;j<theElp->nnds;j++)
            np[j] = theElp->nps[j];
        for(i=0;i<ni;i++) {
            get_shape(theElp->vtype,&g[i],&bf) ;
            get_shape(theElp->gtype,&g[i],&tf) ;
            get_PGgf(theElp,&gbf,&gtf,&bf,&tf,np,g[i].w) ;
            if(tvals.method ==5)
                update_TGKe(&gbf,&gtf,nbf,*ntf,np,nvar,theElp) ;
            else
                update_PGKe(&gbf,&gtf,nbf,*ntf,np,nvar,theElp) ;
        }
        if(code != 2 && N.nonlin == 0) {
            if(elmntp->matrices != NULL)
                free(elmntp->matrices) ;
            nm = *ntf  * (nbf*n + 1) *n ;
            if(N.trans > 0)
                nm += *ntf*n * nbf*n ;
            if((elmntp->matrices = (double *)calloc(nm ,sizeof(double)) ) == NULL)
                return(-1) ;
            fptr = elmntp->matrices ;
            for(i=0;i<*ntf*n;i++)
                for(j=0;j<nbf*n;j++) {
                    *fptr = KE(i,j) ;
                    fptr++ ;
                }
            for(i=0;i<*ntf*n;i++){
                *fptr = FE(i);
                fptr++ ;
            }
            if(N.trans > 0) {
                for(i=0;i<*ntf*n;i++) {
                    for(j=0;j<nbf*n;j++) {
                        *fptr = SE(i,j) ;
                        fptr++ ;
                    }
                }
            }
        }
    }
    else {
        fptr = elmntp->matrices ;
        for(i=0;i<*ntf*n;i++)
            for(j=0;j<nbf*n;j++) {
                KE(i,j) = *fptr ;
                fptr++ ;
            }
        for(i=0;i<*ntf*n;i++) {
            FE(i) = *fptr ;
            fptr++ ;
        }
        if(N.trans > 0) {
            for(i=0;i<*ntf*n;i++) {
                for(j=0;j<nbf*n;j++) {
                    SE(i,j) = *fptr ;
                    fptr++ ;
                }
            }
        }
    }
    if(N.trans > 0) {

    /* mass matrix lumping */
/*
        for(i=0;i<(*ntf)*n;i++) {
            sdiag = 0.0 ;
            for(j=0;j<(nbf)*n;j++) {
                sdiag += SE(i,j) ;
                SE(i,j) = 0.0 ;
            }
            SE(i,i) = sdiag ;
        }
*/
        tdt = tvals.theta * tvals.dt ;
        mtdt = tvals.dt * (1.0 - tvals.theta) ;
        for(i=0;i<(*ntf)*n;i++) {
            FE(i) *= tvals.dt ;
            for(j=0;j<nbf;j++) {
                for(jj=0;jj<n;jj++) {

                    if(itnum == 0)
                        FE(i) += (SE(i,j*n+jj) - mtdt *  KE(i,j*n+jj)) *
                            (theElp->nps[j])->u[jj] ;
                    SE(i,j*n+jj) += tdt * KE(i,j*n+jj) ;
                }
            }
        }
    }
    if(itnum == 0 && N.nonlin > 0) {
        if(elmntp->matrices != NULL)
            free(elmntp->matrices) ;
        nm = *ntf *n ;
        if((elmntp->matrices = (double *)calloc(nm ,sizeof(double)) ) == NULL)
            return(-1) ;
        fptr = elmntp->matrices ;
        for(i=0;i<*ntf*n;i++){
            *fptr = FE(i);
            fptr++ ;
        }
    }
    else {
        fptr = elmntp->matrices ;
        for(i=0;i<*ntf*n;i++) {
            FE(i) = *fptr ;
            fptr++ ;
        }
    }

    if(N.nonlin > 0) {
        for(i=0;i<(*ntf)*n;i++) {
            for(j=0;j<nbf;j++) {
                for(jj=0;jj<n;jj++) {
                    FE(i) -= (SE(i,j*n+jj) * (theElp->nps[j])->u[jj]) ;
                    SE(i,j*n+jj) += tdt * JE(i,j*n+jj) ;
                }
            }
        }
    }
    return(nbf) ;
}

int       ElPick(elmntp,theElp,ntfp,dirp)
struct    element  *elmntp, *theElp ;
int       *ntfp, *dirp ;

{
    int  j;

    *dirp = 1 ;
    if(elmntp->vtype == 129)
        return(QU1DPick(elmntp,theElp,ntfp,dirp)) ;
    else if(elmntp->gtype == 119)
        return(SU1DPick(elmntp,theElp,ntfp,dirp)) ;
    else if(elmntp->vtype == 139)
        return(CU1DPick(elmntp,theElp,ntfp,dirp)) ;
    else if(elmntp->vtype == 229)
        return(QUelPick(elmntp,theElp,ntfp,dirp)) ;
    *ntfp = nsf(elmntp->gtype) ;
    *theElp = *elmntp ;
    return(0) ;

int       get_PGgf(elp,gbfp,gtfp,bfp,tfp,np,w)
struct element          *elp ;
struct shapefuncs       *bfp, *tfp ;
struct node             *np[] ;
struct govfuncs     *gbfp,*gtfp ;
double                  w ;

{
    double   J[NDIMS][NDIMS], get_J() ;
    int      i, j ;

    gbfp->dof = bfp->dof ;
    gtfp->dof = tfp->dof ;

    for(i=0;i<N.params;i++) {
        gbfp->p[i] = elp->p[i] ;
    }
            i = (gbfp->dof > gtfp->dof) ? gbfp->dof : gtfp->dof ;
            for(j=0;j<i;j++) {
                gbfp->f[j] = 0.0  ;
                gtfp->f[j] = 0.0  ;
                gbfp->dfdx[j] = 0.0 ;
                gtfp->dfdx[j] = 0.0 ;
                gbfp->dfdy[j] = 0.0 ;
                gtfp->dfdy[j] = 0.0 ;
            }
    for(j=0;j<bfp->dof;j++)
        gbfp->f[j] = bfp->f[j]   ;
    for(j=0;j<tfp->dof;j++)
        gtfp->f[j] = tfp->f[j]   ;

    gbfp->detJ = get_J(np,bfp,J) ;
    gbfp->w = w ;
    invertJ(J,gbfp->detJ) ;

    for(j=0;j<bfp->dof;j++) {

        switch (N.dims) {

            case 1 :
                gbfp->dfdx[j] = J[0][0] * bfp->dfdr[j] ;
                gbfp->d2fdx[j] = J[0][0] * J[0][0] * bfp->d2fdr[j] ;
                gbfp->dfdy[j] = 0.0 ;
                gbfp->dfdz[j] = 0.0 ;
                break ;

            case 2 :
                gbfp->dfdx[j] = J[0][0] * bfp->dfdr[j] + J[0][1] *bfp->dfds[j] ;
                gbfp->dfdy[j] = J[1][0] * bfp->dfdr[j] + J[1][1] *bfp->dfds[j] ;
                gbfp->dfdz[j] = 0.0 ;
                break ;

            case 3 :
                gbfp->dfdx[j] = J[0][0] * bfp->dfdr[j] + J[0][1] *bfp->dfds[j] + J[0][2] *bfp->dfdt[j] ;
                gbfp->dfdy[j] = J[1][0] * bfp->dfdr[j] + J[1][1] *bfp->dfds[j] + J[1][2] *bfp->dfdt[j] ;
                gbfp->dfdz[j] = J[2][0] * bfp->dfdr[j] + J[2][1] *bfp->dfds[j] + J[2][2] *bfp->dfdt[j] ;
                break ;
        }
    }

    for(j=0;j<tfp->dof;j++) {

        switch (N.dims) {

            case 1 :
                gtfp->dfdx[j] = J[0][0] * tfp->dfdr[j] ;
                gtfp->dfdy[j] = 0.0 ;
                gtfp->dfdz[j] = 0.0 ;
                break ;

            case 2 :
                gtfp->dfdx[j] = J[0][0] * tfp->dfdr[j] + J[0][1] *tfp->dfds[j] ;
                gtfp->dfdy[j] = J[1][0] * tfp->dfdr[j] + J[1][1] *tfp->dfds[j] ;
                gtfp->dfdz[j] = 0.0 ;
                break ;

            case 3 :
                gtfp->dfdx[j] = J[0][0] * tfp->dfdr[j] + J[0][1] *tfp->dfds[j] + J[0][2] *tfp->dfdt[j] ;
                gtfp->dfdy[j] = J[1][0] * tfp->dfdr[j] + J[1][1] *tfp->dfds[j] + J[1][2] *tfp->dfdt[j] ;
                gtfp->dfdz[j] = J[2][0] * tfp->dfdr[j] + J[2][1] *tfp->dfds[j] + J[2][2] *tfp->dfdt[j] ;
                break ;
        }
    }
    gbfp->detJ *= gbfp->w ;
    return(0) ;
}
```

fecom.c

```
#include "fe.h"

#define  KE(A,B) ( *(ep.K + ep.n*(A) + (B)) )
#define  SE(A,B) ( *(ep.S + ep.n*(A) + (B)) )
#define  ME(A,B) ( *(ep.M + ep.n*(A) + (B)) )
#define  JE(A,B) ( *(ep.J + ep.n*(A) + (B)) )
#define  FE(B)     ( *(ep.F + B) )

extern struct control       N ;
extern struct pointers      gp;
extern struct RegMesh       Mesh ;
extern struct fxnodes       *first_fnp ;
extern int                  Nukns ;
extern struct elmpointers   ep ;
extern struct transient     tvals ;
extern struct eqnset        eqnsets[4] ;
extern double               uchange ;


int  varindex(ndindex,var)
int      ndindex, var ;

{
     int  i, vi ;

     vi = gp.iptrs[ndindex]->ui ;
     for(i=0;i<var;i++) {
          if(isvarfixed(ndindex,i) == 0)
               vi++ ;
     }
     return(vi) ;
}

int  isvarfixed(ndindex,var)
int      ndindex, var ;

{
     int  fixcode, rc ;

     rc = 0 ;
     fixcode = (1 << var) ;
     if( (fixcode & gp.iptrs[ndindex]->fxc) != 0)
          rc = -1 ;
     return(rc) ;
}

int  one_step(nvar)
int      nvar ;

{
     int      i ;

     for(i=0;i<tvals.nsteps;i++) {
          tvals.t += tvals.dt ;
          assemble(nvar,0,0,0) ;
          solve(nvar,1,0) ;
          tvals.dt = tvals.dt * tvals.dtfac ;
     }
     return(0) ;
}

int  solve(nvar,code,nset)
int      code, nvar, nset ;

{
     int  i, j, ind, solvecode ;
     double   *dp, varsum ;

     if(code > 1)
          solvecode = 1 ;
     else
          solvecode = code ;
     if(code < 0) {
          solvecode = 2 ;
          code = - code ;
     }
```

```
/*
    printf(" Now solving ...") ;
*/
    if((N.sym == 0) || (code == 3))
        actcol(eqnsets[nset].Kp,eqnsets[nset].Fp,eqnsets[nset].diagp,eqnsets[nset].neqns, solvecode) ;
    else
        uactcl(eqnsets[nset].Kp,eqnsets[nset].Lp,eqnsets[nset].Fp,eqnsets[nset].diagp, eqnsets[nset].neqns,
            solvecode) ;
/*
    printf(" All done !\n") ;
*/

    dp = eqnsets[nset].Fp ;
    if((code > 0) && (code != 3)) {
        if(nvar >= 0) {
            ind = 0 ;
            uchange = 0.0 ;
            for(i=0;i<N.nodes;i++) {
                if(code == 1) {
                    uchange += fabs(gp.iptrs[i]->u[nvar] - *dp) ;
                    gp.iptrs[i]->u[nvar] = *dp ;
                }
                if(code == 2) {
                    *(eqnsets[nset+1].Fp + ind) = *(eqnsets[nset].Fp + gp.iptrs[i]->i) ;
                /*
                    gp.iptrs[i]->p[4] = *(eqnsets[nset].Fp + gp.iptrs[i]->i);
                */
                }
                if(code == 4) {
                /*
                    printf(" %f  %f\n",gp.iptrs[i]->u[nvar], *dp);
                */
                    gp.iptrs[i]->u[nvar] += *dp ;
                }
                dp++;
                ind++ ;
            }
        }
        else {
            uchange = 0.0 ;
            varsum = 0.0 ;
            for(i=0;i<N.nodes;i++) {
                for(j=0;j<N.vars;j++) {
                    if(code == 1)
                        gp.iptrs[i]->u[j] = *dp;
                    if(code == 2) {
                        uchange += *dp * *dp;
                        gp.iptrs[i]->u[j] += *dp ;
                        varsum += gp.iptrs[i]->u[j] * gp.iptrs[i]->u[j] ;
                    }
                    dp++;
                }
            }
        uchange = sqrt (uchange/varsum) ;
        }
    }
    return(0) ;
}

int     actcol(Kp,Fp,diagp,neqns,code)
double  *Kp, *Fp ;
long        *diagp ;
int     neqns, code ;

{
    int     i, j, nf, id, n, ni, nj ;
    double  dot(), *ip, *idp, t ;
    char        m1[255], m2[255], m3[255], m4[255] ;

/*
    if(code < 2) {
        sprintf(m1," Now factoring equation # ") ;
        sprintf(m2," 0 ") ;
        sprintf(m3," ") ;
        sprintf(m4," ") ;
        StartMes(m1,m2,m3,m4) ;
    }
*/
    for(j=1;j<neqns;j++) {
/*
        if( (j % 10 == 0) && (code < 2)) {
            sprintf(m2,"  %d ",j) ;
            UpDateMes(m1,m2,m3,m4) ;
        }
*/
        nf = *(diagp+j) - *(diagp+j-1) ;
        if(code < 2) {
            ip = Kp + *(diagp+j-1) + 1 ;
            id = j - nf + 1 ;

            for(i=1;i<nf;i++) {
                n = ((nj = i-1) < (ni = *(diagp+id) - *(diagp+id-1) - 1)) ? nj : ni ;
                idp = Kp + *(diagp+id++) ;
                *ip++ -= dot(ip-n-1,idp-n,n) ;
            }
            ip = Kp + *(diagp+j-1) + 1 ;
            id = j - nf + 1 ;
            idp = Kp + *(diagp+j) ;
            for(i=1;i<nf;i++) {
                t = *ip ;
                *ip /= *(Kp + *(diagp+id++)) ;
                *idp -= t * *ip++ ;
            }
        }
        ip = Kp + *(diagp+j-1) + 1 ;
        id = j - nf + 1 ;
        if(code > 0)
            *(Fp + j) -= dot(ip,Fp+id,nf-1) ;
    }
    if(code == 0) {
        EndMes() ;
        return(0) ;
    }
    for(j=0;j<neqns;j++)
        *(Fp + j) /= *(Kp + *(diagp+j)) ;

    for(j=neqns-1;j>-1;j--) {
        if(j == 0)
            nf = 1;
        else
            nf = *(diagp+j) - *(diagp+j-1) ;
        for(i=1;i<nf;i++) {
            *(Fp + j - nf + i) -= *(Fp + j) * *(Kp + *(diagp+j-1) + i) ;
        }
    }

    if(code < 2)
        EndMes() ;
    return(0) ;
}

int     uacfcl(Kp,Lp,Fp,diagp,neqns,code) ;
double  *Kp, *Lp, *Fp ;
long    *diagp ;
int     neqns, code ;

{
    int         i, j, nf, id, n, ni, nj ;
    double      dot(), *ip, *iLp, *idp, t ;
    char        m1[255], m2[255], m3[255], m4[255] ;

/*  if(code < 2) {
        sprintf(m1," Now factoring equation # ") ;
        sprintf(m2," 0 ") ;
        sprintf(m3," ") ;
        sprintf(m4," ") ;
        StartMes(m1,m2,m3,m4) ;
    }
*/
    for(j=1;j<neqns;j++) {
/*
        if( (j % 10 == 0) && (code < 2)) {
            sprintf(m2,"  %d ",j) ;
            UpDateMes(m1,m2,m3,m4) ;
        }
*/
        nf = *(diagp+j) - *(diagp+j-1) ;
        if(code < 2) {
            iLp = Lp + *(diagp+j-1) + 1 ;
            id = j - nf + 1 ;

            for(i=0;i<nf-1;i++) {
                n = ((nj = i) < (ni = *(diagp+id) - *(diagp+id-1) - 1)) ? nj : ni ;
                idp = Kp + *(diagp+id) ;
                id++ ;
                *ilp = (*ilp - dot(ilp-n,idp-n,n))/ *idp ;
                ilp++ ;
            }
            lp = Kp + *(diagp+j-1) + 1 ;
            id = j - nf + 1 ;

            for(i=0;i<nf;i++) {
                n = ((nj = i) < (ni = *(diagp+id) - *(diagp+id-1) - 1)) ? nj : ni ;
                idp = Lp + *(diagp+id) ;
                id++ ;
                *lp -= dot(lp-n,idp-n,n) ;
                lp++ ;
            }
        }
        ilp = Lp + *(diagp+j-1) + 1 ;
        id = j - nf + 1 ;
        if(code > 0)
            *(Fp + j) -= dot(ilp,Fp+id,nf-1) ;
    }
    if(code == 0) {
        EndMes() ;
        return(0) ;
    }
    for(j=neqns-1;j>-1;j--) {
        *(Fp + j) /= *(Kp + *(diagp+j)) ;
        if(j == 0)
            nf = 1;
        else
            nf = *(diagp+j) - *(diagp+j-1) ;
        for(i=1;i<nf;i++) {
            *(Fp + j - nf + i) -= *(Fp + j) * *(Kp + *(diagp+j-1) + i) ;
        }
    }
    if(code < 2)
        EndMes() ;
    return(0) ;
}


double   dot(p1,p2,n)
double   *p1, *p2 ;
int      n ;

{
    register double   t=0.0 ;

    while(n-- > 0) {
            t += *p1++ * *p2++ ;
    }
    return(t) ;
}

int        get_Derivs(elmntp,pt,nvar,ddx,ddy,ddz)
struct element          *elmntp ;
struct gausspts         *pt ;
int                     nvar ;
double                  *ddx, *ddy, *ddz ;

{
    struct shapefuncs       fv;
    struct node             *np[MNSF] ;
    struct govfuncs         gf ;
    int                     k, j, ns, nl, nf ;

    ns =  nsf(elmntp->vtype) ;
    for(j=0;j<elmntp->nnds;j++)
        np[j] = elmntp->nps[j] ;
    get_shape(elmntp->vtype,pt,&fv) ;
    get_gf(elmntp,&gf,&fv,&fv,np,pt->w) ;
    *ddx = *ddy = *ddz = 0.0 ;
    for(j=0;j<elmntp->nnds;j++) {
        *ddx += gf.dfdx[j] * np[j]->u[nvar] ;
        *ddy += gf.dfdy[j] * np[j]->u[nvar] ;
        *ddz += gf.dfdz[j] * np[j]->u[nvar] ;
    }
    *ddx *= gf.p[0] ;
    *ddy *= gf.p[0] ;
    *ddz *= gf.p[0] ;
    return(ns) ;
}

int       get_gf(elp,gfp,fvp,fgp,np,w)
struct element          *elp ;
struct shapefuncs       *fvp, *fgp ;
struct node             *np[MNSF] ;
struct govfuncs         *gfp ;
double                  w ;

{
    double   J[NDIMS][NDIMS], get_J() ;
    int      i, j ;

    gfp->dof = fvp->dof ;

    for(i=0;i<N.params;i++) {
        gfp->p[i] = elp->p[i] ;
        for(j=0;j<fvp->dof;j++)
            gfp->p[i] += fvp->f[j] * np[j]->p[i] ;
    }
    for(j=0;j<fvp->dof;j++)
        gfp->f[j] = fvp->f[j]  ;

    gfp->detJ = get_J(np,fgp,J) ;
    gfp->w = w ;
    invertJ(J,gfp->detJ) ;

    for(j=0;j<fvp->dof;j++) {

        switch (N.dims) {

            case 1 :
                gfp->dfdx[j] = J[0][0] * fvp->dfdr[j] ;
                gfp->dfdy[j] = 0.0 ;
                gfp->dfdz[j] = 0.0 ;
                break ;

            case 2 :
                gfp->dfdx[j] = J[0][0] * fvp->dfdr[j] + J[0][1] * fvp->dfds[j] ;
                gfp->dfdy[j] = J[1][0] * fvp->dfdr[j] + J[1][1] * fvp->dfds[j] ;
                gfp->dfdz[j] = 0.0 ;
                break ;

            case 3 :
                gfp->dfdx[j] = J[0][0] * fvp->dfdr[j] + J[0][1] * fvp->dfds[j] + J[0][2] *fvp->dfdt[j] ;
                gfp->dfdy[j] = J[1][0] * fvp->dfdr[j] + J[1][1] * fvp->dfds[j] + J[1][2] *fvp->dfdt[j] ;
                gfp->dfdz[j] = J[2][0] * fvp->dfdr[j] + J[2][1] * fvp->dfds[j] + J[2][2] *fvp->dfdt[j] ;
                break ;
        }
    }
    gfp->detJ *= gfp->w ;
    return(0) ;
}

double    get_J(np,fgp,J)
struct shapefuncs       *fgp ;
struct node             *np[MNSF] ;
double                  J[NDIMS][NDIMS] ;

{
    int  i,j ;

    for(i=0;i<N.dims;i++)
        for(j=0;j<N.dims;j++)
            J[i][j] = 0.0 ;

    for(i=0;i<fgp->dof;i++)
        J[0][0] += fgp->dfdr[i] * np[i]->x[0] ;
    if(N.dims == 1)
        return( J[0][0] ) ;

    for(i=0;i<fgp->dof;i++) {
        J[0][1] += fgp->dfdr[i] * np[i]->x[1] ;
        J[1][0] += fgp->dfds[i] * np[i]->x[0] ;
        J[1][1] += fgp->dfds[i] * np[i]->x[1] ;
    }
    if(N.dims == 2)
        return( J[0][0] * J[1][1] - J[0][1] * J[1][0] ) ;

    for(i=0;i<fgp->dof;i++) {
        J[0][2] += fgp->dfdr[i] * np[i]->x[2] ;
        J[1][2] += fgp->dfds[i] * np[i]->x[2] ;
        J[2][2] += fgp->dfdt[i] * np[i]->x[2] ;
        J[2][0] += fgp->dfdt[i] * np[i]->x[0] ;
        J[2][1] += fgp->dfdt[i] * np[i]->x[1] ;
    }
    return( J[0][0] * (J[1][1] * J[2][2] - J[1][2] * J[2][1] )
        -  J[0][1] * (J[1][0] * J[2][2] - J[2][0] * J[1][2] )
        +  J[0][2] * (J[1][0] * J[2][1] - J[1][1] * J[2][1] ) ) ;
}

int        invertJ(J,detJ)
double     J[NDIMS][NDIMS], detJ ;

{
    double    K[NDIMS][NDIMS] ;
    int       i,j ;

    for(i=0;i<N.dims;i++)
        for(j=0;j<N.dims;j++)
            K[i][j] = J[i][j] ;

    switch (N.dims) {

        case 1 :
            J[0][0] = 1.0 / K[0][0] ;
            break ;

        case 2 :
            J[0][0] =  K[1][1] / detJ ;
            J[0][1] = -K[0][1] / detJ ;
            J[1][0] = -K[1][0] / detJ ;
            J[1][1] =  K[0][0] / detJ ;
            break ;

        case 3 :
            J[0][0] = ( K[1][1] * K[2][2] - K[1][2] * K[2][1] ) / detJ ;
            J[0][1] = ( K[0][2] * K[2][1] - K[0][1] * K[2][2] ) / detJ ;
            J[0][2] = ( K[0][1] * K[1][2] - K[1][1] * K[0][2] ) / detJ ;
            J[1][0] = ( K[1][2] * K[2][0] - K[1][0] * K[2][2] ) / detJ ;
            J[1][1] = ( K[0][0] * K[2][2] - K[0][2] * K[2][0] ) / detJ ;
            J[1][2] = ( K[1][0] * K[0][2] - K[0][0] * K[1][2] ) / detJ ;
            J[2][0] = ( K[1][0] * K[2][1] - K[1][1] * K[2][0] ) / detJ ;
            J[2][1] = ( K[0][1] * K[2][0] - K[0][0] * K[2][1] ) / detJ ;
            J[2][2] = ( K[0][0] * K[1][1] - K[0][1] * K[1][0] ) / detJ ;
            break ;
    }
    return(0) ;
}


int        get_bKe(elmntp,theElp,nvar,code,ntf,itnum)
struct belement    *elmntp, *theElp ;
int                nvar, code, *ntf, itnum ;

{

    struct gausspts        g[MGPTS] ;
    struct shapefuncs      fv, ft, *fgp ;
    struct node            *np[MNSF] ;
    struct govfuncs        gf, gtf ;
    int                    i, j, ig, nbf, ni, nm, dir, n, jj ;
    double                 *fptr ;
    double                 J00, J01, dS, tdt, mtdt ;

    if(nvar < 0) {
        n = N.vars ;
    }
    else {
        n = 1 ;
    }
    nbf =  nsf(elmntp->vtype) ;
    *ntf = nsf(elmntp->gtype) ;
    initm(ep.K,ep.n*ep.n) ;
    if(N.trans > 0)
        initm(ep.S,ep.n*ep.n) ;
    if(N.nonlin > 0)
        initm(ep.J,ep.n*ep.n) ;
    initm(ep.F,ep.n) ;
    *theElp = *elmntp ;
    if(code > -1 ) {
        ni = get_gspts(elmntp->vtype,g) ;
        for(j=0;j<elmntp->nnds;j++)
            np[j] = elmntp->nps[j] ;

        for(ig=0;ig<ni;ig++) {
            get_shape(elmntp->vtype,&g[ig],&fv) ;
            get_shape(elmntp->gtype,&g[ig],&ft) ;

            J00 = J01 = 0.0 ;
            fgp = &fv ;
            for(i=0;i<fgp->dof;i++) {
                J00 += fgp->dfdr[i] * np[i]->x[0] ;
                J01 += fgp->dfdr[i] * np[i]->x[1] ;
            }
            dS = g[ig].w * sqrt(J00 * J00 + J01 * J01) ;
            update_BKe(elmntp,&fv,&ft,nbf,*ntf,np,nvar,dS) ;
        }
    }
    if(N.trans > 0) {
        tdt = tvals.theta * tvals.dt ;
        mtdt = tvals.dt * (1.0 - tvals.theta) ;
        for(i=0;i<*(ntf)*n;i++) {
            FE(i) *= tvals.dt ;
            for(j=0;j<nbf;j++) {
                    for(jj=0;jj<n;jj++) {

                    if(itnum == 0)
                        FE(i) += (SE(i,j*n+jj) - mtdt *  KE(i,j*n+jj)) * (theElp->nps[j])->u[jj] ;

                    SE(i,j*n+jj) += tdt * KE(i,j*n+jj) ;
                }
            }

        }
    }
    if(itnum == 0 && N.nonlin > 0) {
        if(elmntp->matrices != NULL)
            free(elmntp->matrices) ;
        nm = *ntf *n ;
        if((elmntp->matrices = (double *)calloc(nm ,sizeof(double)) ) == NULL)
            return(-1) ;
        fptr = elmntp->matrices ;
        for(i=0;i<*ntf*n;i++){
            *fptr = FE(i);
            fptr++ ;
        }
    }
    else {
        fptr = elmntp->matrices ;
        for(i=0;i<*ntf*n;i++) {
            FE(i) = *fptr ;
            fptr++ ;
        }
    }

    if(N.nonlin > 0) {
        for(i=0;i<(*ntf)*n;i++) {
            for(j=0;j<nbf;j++) {
                for(jj=0;jj<n;jj++) {
                    FE(i) -= (SE(i,j*n+jj) * (theElp->nps[j])->u[jj]) ;
                    SE(i,j*n+jj) += tdt * JE(i,j*n+jj) ;
                }
            }
        }
    }

    return(nbf) ;
}


int       initm(K,n)
double    *K ;
int       n ;

{

    register int  i ;

    for(i=0;i<n;i++)
            *(K++) = 0.0 ;
    return(0) ;
```

```
}

double          GetElErr(elmntp,nvar)
struct element              *elmntp ;
int                         nvar ;

{
/*
    struct gausspts           g[MGPTS] ;
    struct shapefuncs         fv, fve, fg, *fgp;
    struct node               *np[MNSF] ;
    struct govfuncs           gf ;
    int                       i, ii, j, ns, ni ;
    double                    J[NDIMS][NDIMS], dfdxe, dfdye,fxiui, fyiui, err, area ;

    ns =  nsf(elmntp->vtype) ;
    ni = get_gspts(elmntp->vtype,g) ;
    for(j=0;j<elmntp->nnds;j++)
        np[j] = elmntp->nps[j]] ;

    err = area = 0.0 ;
    for(i=0;i<ni;i++) {
        get_shape(elmntp->vtype,&g[i],&fv) ;
        mid_pt(&g[i],&fve) ;
        fgp = &fv ;
        get_gf(elmntp,&gf,&fv,fgp,np,g[i].w) ;

        for(ii=0;ii<N.dims;ii++)
            for(j=0;j<N.dims;j++)
                J[ii][j] = 0.0 ;

        for(ii=0;ii<fgp->dof;ii++) {
            J[0][0] += fv.dfdr[ii] * np[ii]->x[0] ;
            J[0][1] += fv.dfdr[ii] * np[ii]->x[1] ;
            J[1][0] += fv.dfds[ii] * np[ii]->x[0] ;
            J[1][1] += fv.dfds[ii] * np[ii]->x[1] ;
        }
        invertJ(J,gf.detJ) ;
        dfdxe = J[0][0] * fve.dfdr[8] + J[0][1] * fve.dfds[8] ;
        dfdye = J[1][0] * fve.dfdr[8] + J[1][1] * fve.dfds[8] ;

        fxiui = fyiui = 0.0 ;
        for(j=0;j<fgp->dof;j++) {
            fxiui += gf.dfdx[j] * np[j]->u[nvar] ;
            fyiui += gf.dfdy[j] * np[j]->u[nvar] ;
        }
        err += (fxiui * dfdxe + fyiui * dfdye) * gf.detJ ;
        area += gf.detJ ;
    }
*/
    return(0.0) ;
}

int mid_pt(x,p)
struct gausspts         *x ;
struct shapefuncs       *p ;

{
    double   r,s ;

    r = x->x ;
    s = x->y ;

    p->dfdr[8] = -2.0 * r * (1.0 - s * s) ;
    p->dfds[8] = -2.0 * s * (1.0 - r * r) ;

    return(0) ;
}

int         NodalFlux(np,nvar,fx,fy,fz)
struct node     *np ;
double          *fx,*fy,*fz ;
int             nvar ;

{
    double   fxe[10], fye[10], fze[10] ;
    int                  i, j, in;
    struct element    *elp ;
    struct gausspts   pt ;

    elp = gp.El ;
*/
```

```
    i = 0 ;
    pt.z = pt.w = 0.0 ;
    while(elp != NULL) {
        in = NdinEl(np->n,&elp) ;
        if(in < 0)
            break ;
        get_nlcs(elp->vtype,in,&(pt.x),&(pt.y)) ;
        get_Derivs(elp,&pt,nvar,&fxe[i],&fye[i],&fze[i]) ;
        i++ ;
        elp = elp->nextelp ;
    }
    *fx = *fy = *fz = 0.0 ;
    for(j=0;j<i;j++) {
        *fx += fxe[j] ;
        *fy += fye[j] ;
        *fz += fze[j] ;
    }
    if(i > 0) {
        *fx /= i ;
        *fy /= i ;
        *fz /= i ;
    }
    return(i) ;
}

int             NdinEl(n,elpp)
int             n ;
struct element  *(*elpp) ;

{
    int             i, rc;
    struct element  *selp ;

    rc = 0 ;
    selp = *elpp ;
    while(selp != NULL) {
        for(i=0;i<selp->nnds;i++) {
            if(n == (selp->nps[i])->n) {
                rc = 1 ;
                *elpp = selp ;
                break ;
            }
        }
        if(rc == 1)
            return(i) ;
        selp = selp->nextelp ;
    }
    return(-1) ;
}


/*

double   Integrate(elcode,elp,nvar)
int             elcode, nvar ;
struct element  *elp ;

{
    struct node         *np[MNSF] ;
    struct gausspts     g[MGPTS] ;
    struct shapefuncs   sf ;
    int                 i, j, ng ;
    double              integral = 0.0 , t[10] ;

    for(i=0;i<elp->nnds;i++)
        np[i] = elp->nps[i] ;
    ng = get_gspts(elp->vtype,g) ;
    for(i=0;i<ng;i++) {
        for(j=0;j<10;j++)
            t[j] = 0.0 ;
        get_shape(elp->vtype,&g[i],&sf) ;
        switch (elcode) {
            case   1 :
                switch (N.dims) {
                    case 2 :
                        for(j=0;j<sf.dof;j++) {
                            t[0] += sf.f[j] * np[j]->u[nvar] ;
                            t[1] += sf.dfdr[j] * np[j]->x[0] ;
                            t[2] += sf.dfdr[j] * np[j]->x[1] ;
                        }
                        integral += t[0] * sqrt(t[1]*t[1] + t[2]*t[2]) * g[i].w ;
                        break ;
                    case 3 :
                        break ;
                }
                break ;
            case 0 :
                break ;
        }
    }
    return(integral) ;
}
*/

int        get_cline(elp,cval,niv,nvar,tol,xl,yl)
int        niv, nvar ;
double     xl[], yl[], tol, cval ;
struct element     *elp ;

{
    int                     i, j, nipts, rc ;
    double                  x, y, Dx, Dy, dfdr, dfds ;
    struct node             *np[MNSF] ;
    struct shapefuncs       sf ;
    struct gausspts         xg ;

    if(niv == 0)
        tol = -1.0 ;
    nipts = isvalinel(elp,np,cval,nvar,tol,&dfdr,&dfds,xl,yl) ;
    switch (nipts) {
        case     0 :
            break ;
        case 2 :
            Dx = xl[1] - xl[0] ;
            Dy =  yl[1] - yl[0] ;
            xl[niv+1] = xl[1] ;
            yl[niv+1] = yl[1] ;
            for(i=1;i<=niv;i++) {
                xl[i] = xl[i-1] + (xl[niv+1] - xl[i-1]) * 1 / (niv - i + 2);
                yl[i] = yl[i-1] + (yl[niv+1] - yl[i-1]) * 1 / (niv - i + 2);
                find_2Dloc(np,nvar+N.dims,elp->gtype,cval,tol,Dx,Dy,dfdr,dfds,&xl[i],&yl[i]) ;
            }
            xg.z = 0.0 ;
            xg.w = -1.0 ;
            for(i=0;i<=niv+1;i++) {
                xg.x = xl[i] ;
                xg.y = yl[i] ;
                rc = get_shape(elp->gtype,&xg,&sf) ;
                xl[i] = yl[i] = 0.0 ;
                for(j=0;j<sf.dof;j++) {
                    xl[i] += sf.f[j] * np[j]->x[0] ;
                    yl[i] += sf.f[j] * np[j]->x[1] ;
                }
            }
            break ;
    }

    return(nipts) ;
}

int        isvalinel(elp,np,cval,nvar,tol,dfdr,dfds,xl,yl)
int                  nvar ;
double               xl[], yl[], tol, cval, *dfdr, *dfds ;
struct node          *np[] ;
struct element       *elp ;

{
    int                 j, k, nbps, etyp, ng, next, prev, ncpts, aj[MNSF], ak[MNSF] ;
    double              Dx, Dy, r, s, t, f, fr, fs, ft ;
    struct gausspts     g[MGPTS] ;

    nbps = nbounds(elp->gtype) ;
    etyp = elp->gtype ;
    for(j=0;j<elp->nnds;j++)
        np[j] =  elp->nps[j] ;
    prev = (cval <= np[nbps-1]->u[nvar]) ? 1 : -1 ;
    ncpts = 0 ;
    for(j=0;j<nbps;j++) {
        next = (cval <= np[j]->u[nvar]) ? 1 : -1 ;
        if(next != prev) {
            aj[ncpts] = j ;
            if( j == 0 )
            ak[ncpts] = nbps-1 ;
        else
            ak[ncpts] = j - 1 ;
        ncpts++ ;
      }
      prev = next ;
  }
  if((ncpts > 0) && (tol > 0.0) ) {
      ng = get_gspts(etyp,g) ;
      *dfdr = *dfds = 0.0 ;
      for(j=0;j<ng;j++) {
          r = g[j].x ;
          s = g[j].y ;
          t = g[j].z ;
          get_value(np,nvar,etyp,r,s,t,2,&f,&fr,&fs,&ft) ;
          *dfdr += fr ;
          *dfds += fs ;
      }
      *dfdr /= ng ;
      *dfds /= ng ;
  }
  for(j=0;j<ncpts;j++) {
          GetIPoint(aj[j],ak[j],np,etyp,cval,nvar,&xl[j],&yl[j],&Dx,&Dy);
          if(tol > 0.0)
              find_2Dloc(np,nvar+N.dims,etyp,cval,tol,Dx,Dy,*dfdr,*dfds,&xl[j],&yl[j]) ;
  }
  return(ncpts) ;
}

int         GetIPoint(j,k,np,eltype,u,n,xp,yp,Dx,Dy)
double          *xp, *yp, *Dx, *Dy, u ;
int             j, k, n, eltype;
struct node     *np[] ;

{
    double   x1, y1, x2, y2 ;

    get_nlcs(eltype,k,&x1,&y1) ;
    get_nlcs(eltype,j,&x2,&y2) ;
    *xp = x1 + (u - np[k]->u[n])/(np[j]->u[n] - np[k]->u[n])*(x2 - x1) ;
    *yp = y1 + (u - np[k]->u[n])/(np[j]->u[n] - np[k]->u[n])*(y2 - y1) ;
    *Dx = y2 - y1 ;
    *Dy = x1 - x2 ;
    return(0) ;
}

int         find_2Dloc(np,nvar,eltype,fc,tol,Dr,Ds,dfdr,dfds,r,s)
double                fc, tol, Dr, Ds, dfdr, dfds, *r, *s ;
int                   eltype, nvar ;
struct node           *np[] ;

{
    int       i, eq, imax = 100 ;
    double    t, f, ddr, dds, dfdt, df, dr, ds, sl ;

    printf(" in find_2Dloc...") ;

    t = 0.0 ;
    if(fabs(Dr) >= fabs(Ds)) {
        eq = 2 ;
        sl = -Ds/Dr ;
    }
    else {
        eq = 1 ;
        sl = -Dr/Ds ;
    }
    for(i=0;i<imax;i++) {
        if( get_value(np,nvar,eltype,*r,*s,t,2,&f,&ddr,&dds,&dfdt)  != 0)
            return(-1);
        if(fabs(df = f - fc) < tol) {
            printf(" fc %f f %f df %f tol %f\n",fc,f,df,tol);
            break ;
        }
        if(eq == 2) {
            ds = df / (ddr*sl + dds) ;
            dr = sl*ds ;
        }
        else {
            dr = df / (ddr + sl*dds) ;
            ds = sl * dr ;
        }
        if(fabs(dr) > 0.5 || fabs(ds) > 0.5) {
            if(eq == 2) {
                ds = df / (dfdr*sl + dfds) ;
                dr = sl*ds ;
            }
            else {
                dr = df / (dfdr + sl*dfds) ;
                ds = sl * dr ;
            }
        }
        if(fabs(dr) > 0.5 || fabs(ds) > 0.5)
            return(-2) ;
        *r -= dr ;
        *s -= ds ;
        printf(" %d ",i) ;
    }
    printf(" ...out\n") ;
    return(i) ;
}

int      get_value(np,nvar,eltype,r,s,t,code,f,dfdr,dfds,dfdt)
double              *f, *dfdr, *dfds, *dfdt, r, s, t ;
int                 eltype, nvar, code ;
struct node         *np[] ;

{
    int                     i, rc ;
    struct shapefuncs       sf ;
    struct gausspts         x ;

    x.x = r ;
    x.y = s ;
    x.z = t ;
    x.w = 0.0  ;
    if(code == 0)
        x.w = -1.0 ;
    rc = get_shape(eltype,&x,&sf) ;

    switch (code) {
        case     3 :
            *dfdt = 0.0 ;
            for(i=0;i<sf.dof;i++)
                *dfdt += sf.dfdt[i] * np[i]->x[nvar] ;
        case 2 :
            *dfds = 0.0 ;
            for(i=0;i<sf.dof;i++)
                *dfds += sf.dfds[i] * np[i]->x[nvar] ;
        case 1 :
            *dfdr = 0.0 ;
            for(i=0;i<sf.dof;i++)
                *dfdr += sf.dfdr[i] * np[i]->x[nvar] ;
        case 0 :
            *f = 0.0 ;
            for(i=0;i<sf.dof;i++)
                *f += sf.f[i] * np[i]->x[nvar] ;
    }
    return(rc) ;
}
```

fegauss.c

```
#include "fe.h"

int                  get_gspts(i_type,g)
int                  i_type ;
struct gausspts      g[] ;

{
    int n ;

    switch (i_type) {

        case 1 :
            n = gs_pt(g) ;
            break ;

        case 111 :
            n = gs_1D2(g) ;
            break ;

        case 119 :
            n = gs_1D3(g) ;
            break ;

        case 118 :
            n = gs_1D3(g) ;
            break ;

        case 128 :
            n = gs_1D2(g) ;
            break ;

        case 129 :
            n = gs_1D3(g) ;
            break ;

        case 121 :
            n = gs_1D3(g) ;
            break ;

        case 131 :
            n = gs_1D3(g) ;
            break ;

        case 138 :
            n = gs_1D3(g) ;
            break ;

        case 139 :
            n = gs_1D3(g) ;
            break ;


        default :
            n = 0 ;
            break ;
    }

    return(n) ;
}

int       gs_pt(g)
struct gausspts          g[] ;

{
    g[0].x =  0.0 ;
    g[0].y =  0.0 ;
    g[0].z =  0.0 ;
    g[0].w =  1.0 ;

    return(1) ;
}

int       gs_1D2(g)
struct gausspts          g[] ;

{
    g[0].x = -0.577350269189626     ;
    g[0].y =  0.0 ;
    g[0].z =  0.0 ;
    g[0].w =  1.0 ;

    g[1].x =  0.577350269189626     ;
    g[1].y =  0.0 ;
    g[1].z =  0.0 ;
    g[1].w =  1.0 ;

    return(2) ;
}

int      gs_1D3(g)
struct gausspts        g[] ;

{
    g[0].x = -0.774596669241483 ;
    g[0].y =  0.0 ;
    g[0].z =  0.0 ;
    g[0].w =  0.555555555555556 ;

    g[1].x =  0.0 ;
    g[1].y =  0.0 ;
    g[1].z =  0.0 ;
    g[1].w =  0.888888888888889 ;

    g[2].x =  0.774596669241483 ;
    g[2].y =  0.0 ;
    g[2].z =  0.0 ;
    g[2].w =  0.555555555555556 ;

    return(3) ;
}
```

fetest.c

```c
#include "fe.h"

#define  KE(A,B)  ( *(ep.K + ep.n*(A) + (B)) )
#define  SE(A,B)  ( *(ep.S + ep.n*(A) + (B)) )
#define  ME(A,B)  ( *(ep.M + ep.n*(A) + (B)) )
#define  FE(B)  ( *(ep.F + B) )
#define       TRUE -1

extern struct control          N ;
extern struct pointers         gp;
extern struct RegMesh          Mesh ;
extern struct fxnodes          *first_fnp ;
extern int                     Nukns ;
extern struct elmpointers      ep ;
extern struct transient        tvals ;
extern struct RegMesh          Mesh ;

int       list_vars(nvar)
int       nvar ;

{
    int       i;
    double    uexact() ;

    printf(" Node\t Value \n\n");
    for(i=0;i<N.nodes;i++) {
        printf(" %d\t %20.14g  %20.14g\n",gp.iptrs[i]->n,gp.iptrs[i]->u[nvar],
            uexact(nvar,gp.iptrs[i]->x[0],0.0,0.0,gp.iptrs[i]->p[2]));
    }
    return ;
}

int       set_param(nparam)
int       nparam ;

{
    int       i;
    double    p ;

    printf(" Input value ");
    scanf(" %lf",&p) ;
    for(i=0;i<N.nodes;i++) {
        gp.iptrs[i]->p[nparam] = p;
    }
    printf(" \n") ;
    return ;
}

int       set_ptou(nparam)
int       nparam ;

{
    int  i;
    double    p ;

    for(i=0;i<N.nodes;i++) {
        update_p(gp.iptrs[i]);
    }
    return ;
}

int       test_elK(nelm,nvar)
int       nelm ;

{
    int       i,j, ii, jj,err, ntf, n, itnum ;
    struct element       *elp, tel ;
        printf("\n Function test_elK \n");

    set_ptou(0) ;
    if(nvar < 0)
        n = N.vars ;
    else
        n = 1 ;
    if( (ep.n = ep_init() ) == 0 )
        return(-1) ;
    elp = gp.El ;
    for(i=0;i<N.elms;i++) {
        if(elp->n == nelm)
            break ;
        elp = elp->nextelp ;
    }
    err = get_PGKe(elp,&tel,-1,2,&ntf,itnum) ;
    printf("\n  K matrix and F vector for element %d\n",nelm) ;
    for(j=0;j<err;j++) {
            printf(" %d",tel.nps[j]->i) ;
    }
    for(i=0;i<ntf;i++) {
        for(ii=0;ii<n;ii++) {
            printf("\n %d ",tel.nps[i]->i) ;
            for(j=0;j<err;j++) {
                for(jj=0;jj<n;jj++)
                    printf(" %9.4f",KE(i*n+ii,j*n+jj)) ;
            }
            printf(" %12.4g",FE(i*n+ii)) ;
        }
    }
    if(N.trans > 0) {
        printf("\n  S matrix  for element %d\n",nelm) ;
        for(i=0;i<ntf;i++) {
            for(ii=0;ii<n;ii++) {
                printf("\n ") ;
                for(j=0;j<err;j++) {
                    for(jj=0;jj<n;jj++)
                        printf(" %9.4f",SE(i*n+ii,j*n+jj)) ;
                }
            }
        }
    }
    printf("\n\n") ;
    return(err) ;
}


int     test_trans(nvar)
int     nvar ;

{
    int  i, nt ;
    double   tol ;

    printf(" Time = %lf \n", tvals.t) ;
    printf(" Input theta, delta t and magnification factor \n") ;
    scanf(" %lf %lf %lf",&tvals.theta,&tvals.dt,&tvals.dtfac) ;
    printf(" Input number of time steps and tolerance \n") ;
    scanf(" %d %lf",&tvals.nsteps,&tol) ;
    printf(" Input method number and and number of steps between printouts\n") ;
    scanf(" %d %d",&tvals.method,&tvals.iter) ;
    transient(nvar,2,tol) ;
    return(0) ;
}

int       test_time(nvar)
int       nvar ;

{

    tvals.t = nvar ;
    printf(" Time = %lf \n", tvals.t) ;
    return(0) ;
}

int       test_ice(nvar)
int       nvar ;

{
    double   xtoe,xlen,kice ;

    printf(" Time = %lf \n", tvals.t) ;
    printf(" Input jam location, length and roughness height \n") ;
    scanf(" %lf %lf %lf",&xtoe,&xlen,&kice) ;
    icejam(xtoe,xlen,kice) ;
    return(0) ;
```

```
{
    struct node         *np0, *npp, *npm ;
    int i, j;

    for(j=0;j<3;j++){
        npm = gp.N ;
        np0 = npm ->nextnp ;
        npp = np0 -> nextnp ;

        for (i=1;i<N.nodes-1;i++){
            np0->x[nvar] = np0->x[nvar] + (npm->x[nvar] - 2*np0->x[nvar] + npp->x[nvar])/2 ;
            npm = np0 ;
            np0 = npp ;
            npp = npp -> nextnp ;
            }
    }
    return(0) ;
}

int         test_mesh(eltype)
int         eltype ;

{
    int         i, ni ;
    struct element      *elp ;

    Mesh.eltype = eltype ;
    printf(" Input nx, nbx, ny, nby \n") ;
    scanf(" %d %d %d %d",&Mesh.nx,&Mesh.nbx,&Mesh.ny,&Mesh.nby) ;
    ni = MkLapMesh(&Mesh) ;
    printf(" return code = %d\n",ni) ;
    return(0) ;
}
```

feinitial.c

```
#include "fe.h"

extern struct control      N ;
extern struct pointers     p;
extern int                 BandWidth ;
extern struct transient    tvals ;
extern struct RegMesh      Mesh ;

int       Nxb, Nyb ;
double    Utop, UW ;


int       localinit()

{
    BandWidth = 4;
    Nxb = 5 ;
    Nyb = 5 ;
    Utop = 1.0 ;
    UW = 0.25 ;

    return(0) ;
}

int       nodevalues(np)
struct node   *np ;

{
    /* x coordinates */
    np->x[0] = np->n * 5.0 ;
    /* y coordinates */
    np->x[1] = (np->i %(Mesh.ny+1)) * 200.0 - 3400.0 ;

    /* phi values - A and Q */
    /*if ((np->n)>=6)
        np->u[0] = 1.0 ;
    else*/
        np->u[0] = 19.0 ;
    np->u[1] = 5.0 ;

    /* parameters */
    /* zmin */
    /*np->p[0] = 1.0 - (np->n * 0.10);*/
    np->p[0] = 0.0 ;
    /* C* */
    np->p[1] = 10.0 ;
    /* b */
    np->p[2] = 10.0 ;

    return(0) ;
}

int       elvalues(elp)
struct element    *elp ;

{
    return(0) ;
}


int       leftb(belp)
struct belement   *belp ;

{
    int scode ;

    belp->bcs[0] = 0 ;
    belp->bcs[1] = 1 ;
    scode = 1.0 ;
    bndparm (belp,scode) ;

    return(0) ;
}
```

```
struct belement   *belp ;

{

    int scode ;


    belp->bcs[0] = 0 ;
    belp->bcs[1] = 1 ;
    scode = -1.0 ;
    bndparm (belp,scodb) ;

    return(0) ;
}


int       bndparm (belp,scode)
struct    belement       *belp ;


{
    belp->p[6] = scode ;

    if (belp->bcs[0] == 0) {
        belp->p[0] = 0.0 ;
        belp->p[1] = 0.0 ;
        belp->p[2] = 0.0 ;
    }
    if (belp->bcs[0] == 1) {
        belp->p[0] = 1.0 ;
        belp->p[1] = 0.0 ;
        belp->p[2] = 1.0 ;
    }
    if (belp->bcs[1] == 0) {
        belp->p[3] = 0.0 ;
        belp->p[4] = 0.0 ;
        belp->p[5] = 0.0 ;
    }
    if (belp->bcs[1] == 1) {
        belp->p[3] = 0.0 ;
        belp->p[4] = 1.0 ;
        belp->p[5] = 1.0 ;
    }
    return(0) ;
}

int       update_p(np)
struct node   *np ;

{
    np->uoo[0] = np->uo[0] ;
    np->uo[0] = np->u[0] ;
    np->uoo[1] = np->uo[1] ;
    np->uo[1] = np->u[1] ;
    return(0) ;
}
```

femesh.c

```
#include "fe.h"
#include "fe.n"
#include "ctype.h"

extern struct control       N ;
extern struct pointers      gp;
extern struct RegMesh       Mesh ;
extern int                  Nndso, Nelmso, Nbelmso ;

int     n1L, n2L ;
double  x0, x1, yzero, yone, L, ar, defndp, defelp ;

int     MakeMesh(theMesh)
struct  RegMesh  *theMesh ;

{
    int             err,i, j, size ;
    long int            bigint ;
    struct node  *anodep ;

    x0 = 0.0 ;
    yzero = 0.0 ;
    x1 = +1.0 + theMesh->nbx * 2.0 ;
    yone = -1.0 + theMesh->nby * 2.0 ;
    n1L = theMesh->ny ;
    n2L = theMesh->nx ;
    L = (yone - yzero)/n1L ;
    ar = (x1 - x0)/(3.0 * n2L) ;
    if(theMesh->eltype == 216 || theMesh->eltype == 226 || theMesh->eltype == 227)
        n2L += 1 ;
    if(gp.N != NULL) {
        free(gp.N) ;
        free(gp.iptrs) ;
        clear_data() ;
    }

    j = 0 ;
    for(i=0;i<1;i++) {
        j++ ;
        switch (theMesh->eltype) {
            case 111 :
                N.nodes = n2L+1 ;
                N.dims = 1 ;
                break ;
            case 119 :
                N.nodes = n2L+1 ;
                N.dims = 1 ;
                break ;
            case 129 :
                N.nodes = n2L + 1;
                N.dims = 1 ;
                break ;
            case 139 :
                N.nodes = n2L + 1;
                N.dims = 1 ;
                break ;
            default :
                return(-1) ;
        }
        bigint = (N.nodes*5)/4 * sizeof(struct node *) ;
        if(N.nodes < 2 || (gp.iptrs = (struct node **) malloc( bigint )) == NULL)
            break ;
        bigint = (long int) N.nodes * (long int) sizeof(struct node) ;
        size = sizeof(struct node) ;
        printf("memory requested = %ld, %d\n",bigint,size) ;
        if(N.nodes < 2 || (gp.N = (struct node *) malloc( bigint )) == NULL)
             break ;
        j++ ;
        N.elms = n1L * n2L ;
        if((theMesh->eltype == 210) || (theMesh->eltype == 220) )
            N.elms *= 2 ;
        bigint = (long int) N.elms * (long int) sizeof(struct element);
        size = sizeof(struct element) ;
        printf("memory requested = %ld, %d\n",bigint,size) ;
        if(gp.El != NULL)
            free(gp.El) ;
        if(N.elms < 1 || (gp.El = (struct element *) malloc( bigint) ) == NULL)
            break ;
        j++ ;
        N.belms = 2 * (n1L + n2L) ;
        if((theMesh->eltype == 111)||(theMesh->eltype == 121)||(theMesh->eltype == 119)||
            (theMesh->eltype == 129)||(theMesh->eltype == 131)||(theMesh->eltype == 139))
                N.belms = 2 ;
        bigint = (long int) N.belms * (long int) sizeof(struct belement) ;
        if(gp.B != NULL)
            free(gp.B) ;
        if(N.belms < 1 || (gp.B = (struct belement *) malloc( bigint )) == NULL)
            break ;
        Nndso = N.nodes ;
        Nelmso = N.elms ;
        Nbelmso = N.belms ;
        put_control(stdout) ;

        for(i=1;i<=n2L;i++)
            for(j=1;j<=n1L;j++)
                make_elmnt(i,j,theMesh->eltype) ;

        set_bvalues() ;
        j = 0 ;
    }
    err = j ;
    if(err != 0)
        printf(" Control Variable Error %d\n",err) ;
    anodep = gp.N ;
    return(err);
}

int        make_elmnt(i,j,eltype)
int        i, j, eltype ;

{
    int   k, index, teltype ;
    double    x, y ;
    struct node  *nodep, *alloc_node() ;
    struct element       *elmntp, *alloc_elm() ;
    struct belement      *belmntp, *alloc_belm() ;

    teltype = eltype ;
    if(eltype == 119)
        teltype = 111 ;
    switch (teltype) {
        case 111 :
            index = i - 1 ;
            x = x0 + j * (i-1) * ar ;
            y = yzero ;
            nodep = alloc_node(index,x,y) ;
            if(i == 1) {
                index = 0 ;
                belmntp = alloc_belm(index,1) ;
                belmntp->nps[0] = gp.N ;
                leftb(belmntp) ;
            }
            if(i == n2L) {
                index = i ;
                x = x1 ;
                y = yzero ;
                nodep = alloc_node(index,x,y) ;
                index = 1 ;
                belmntp = alloc_belm(index,1) ;
                belmntp->nps[0] = gp.N + i ;
                rightb(belmntp) ;
                belmntp->nextbelp = NULL ;
            }
            index = i - 1 ;
            elmntp = alloc_elm(index,eltype) ;
            if(eltype == 119){
                elmntp->vtype = 111;
                elmntp->gtype = 119;
            }
            elmntp->nps[0] = gp.N + i-1 ;
            elmntp->nps[1] = gp.N + i ;
            if(index == N.elms-1)
                elmntp->nextelp = NULL ;
            break ;
        case [illegible]
    index = i - 1 ;
    x = x0 + 3 * (i-1) * ar ;
    y = yzero ;
    nodep = alloc_node(index,x,y) ;
    if(i == 1) {
        index = 0 ;
        belmntp = alloc_belm(index,1) ;
        belmntp->nps[0] = gp.N ;
        leftb(belmntp) ;
    }
    if(i == n2L) {
        index = 1 ;
        x = x1 ;
        y = yzero ;
        nodep = alloc_node(index,x,y) ;
        index = 1 ;
        belmntp = alloc_belm(index,1) ;
        belmntp->nps[0] = gp.N + 1 ;
        rightb(belmntp) ;
        belmntp->nextbelp = NULL ;
    }
    index = i - 1 ;
    elmntp = alloc_elm(index,eltype) ;
    elmntp->nps[0] = gp.N + i-1 ;
    elmntp->nps[1] = gp.N + i ;
    if(i != n2L)
        elmntp->nps[3] = gp.N + i + 1 ;
    else
        elmntp->nps[3] = NULL ;
    if(i != 1)
        elmntp->nps[2] = gp.N + i - 2 ;
    else
        elmntp->nps[2] = NULL ;
    if(index == N.elms-1)
        elmntp->nextelp = NULL ;
    break ;

case 139 :
    index = i - 1 ;
    x = x0 + 3 * (i-1) * ar ;
    y = yzero ;
    nodep = alloc_node(index,x,y) ;
    if(i == 1) {
        index = 0 ;
        belmntp = alloc_belm(index,1) ;
        belmntp->nps[0] = gp.N ;
        leftb(belmntp) ;
    }
    if(i == n2L) {
        index = 1 ;
        x = x1 ;
        y = yzero ;
        nodep = alloc_node(index,x,y) ;
        index = 1 ;
        belmntp = alloc_belm(index,1) ;
        belmntp->nps[0] = gp.N + 1 ;
        rightb(belmntp) ;
        belmntp->nextbelp = NULL ;
    }
    index = i - 1 ;
    elmntp = alloc_elm(index,eltype) ;
    elmntp->nps[0] = gp.N + i-1 ;
    elmntp->nps[1] = gp.N + i ;
    if(i != n2L)
        elmntp->nps[3] = gp.N + i + 1 ;
    else
        elmntp->nps[3] = NULL ;
    if(i == 1) {
        elmntp->nps[2] = NULL ;
        elmntp->nps[3] = NULL ;
    }
    else if(i == 2) {
        elmntp->nps[2] = gp.N + i - 2 ;
        elmntp->nps[3] = NULL ;
    }
    else  {
        elmntp->nps[2] = gp.N + i - 2 ;
        elmntp->nps[3] = gp.N + i - 3  ;
    }
    if(i == n2L) {
        elmntp->nps[4] = NULL ;
        elmntp->nps[5] = NULL ;
    }
    else if(i == n2L-1) {
        elmntp->nps[4] = gp.N + i + 1 ;
        elmntp->nps[5] = NULL ;
    }
    else  {
        elmntp->nps[4] = gp.N + i + 1 ;
        elmntp->nps[5] = gp.N + i + 2  ;
    }

    if(index == N.elms-1)
        elmntp->nextelp = NULL ;
    break ;

case 211 :
    index = (i-1) * (n1L+1) + j - 1 ;
    if(i == 1)
        x = x0 ;
    else
        x = x0 + 3 * (i-1) * ar ;
    y = yzero + (j-1) * L ;
    nodep = alloc_node(index,x,y) ;
    if(j == 1) {
        index = i - 1 ;
        belmntp = alloc_belm(index,111) ;
        belmntp->nps[0] = gp.N + (i-1) * (n1L + 1)  ;
        belmntp->nps[1] = gp.N + i * (n1L + 1)  ;
        bottomb(belmntp) ;
    }
    if(i == 1) {
        index = 2 * n2L + 2 * n1L - j ;
        belmntp = alloc_belm(index,111) ;
        belmntp->nps[0] = gp.N + j ;
        belmntp->nps[1] = gp.N + j - 1 ;
        leftb(belmntp) ;
        if(j == 1)
            belmntp->nextbelp = NULL ;
    }
    if(j == n1L) {
        index = (i-1) * (n1L+1) + j ;
        y = yone ;
        nodep = alloc_node(index,x,y) ;
        nodep->u[1] = 1.0 ;
        index = 2 * n2L + n1L - i ;
        belmntp = alloc_belm(index,111) ;
        belmntp->nps[0] = gp.N + (i + 1 ) * (n1L + 1) -1 ;
        belmntp->nps[1] =  gp.N + i  * (n1L + 1) -1 ;
        topb(belmntp) ;
    }
    if(i == n2L) {
        index = i * (n1L+1) + j - 1 ;
        x = x1 ;
        y = yzero + (j-1) * L ;
        nodep = alloc_node(index,x,y) ;
        index = n2L + j - 1 ;
        belmntp = alloc_belm(index,111) ;
        belmntp->nps[0] = gp.N + n2L * (n1L + 1) + j -1 ;
        belmntp->nps[1] = gp.N + n2L * (n1L + 1) + j ;
        rightb(belmntp) ;
    }
    if(i == n2L && j == n1L) {
        index = i * (n1L+1) + j ;
        x = x1 ;
        y = yone ;
        nodep = alloc_node(index,x,y) ;
    }
    if(eltype == 211) {
        index = (i - 1) * n1L + j - 1;
        elmntp = alloc_elm(index,eltype) ;
        elmntp->nps[3] = gp.N + (i-1) * (n1L + 1) + j -1 ;
        elmntp->nps[0] = gp.N + i * (n1L + 1) + j-1 ;
        elmntp->nps[1] = gp.N + i * (n1L + 1) + j ;
        elmntp->nps[2] = gp.N + (i-1) * (n1L + 1) + j ;
    }
    else {
        index = 2 * ((i - 1) * n1L + j - 1) ;
        elmntp = alloc_elm(index,eltype) ;
        elmntp->nps[0] = gp.N + (i-1) * (n1L + 1) + j - 1 ;
        elmntp->nps[1] = gp.N + i * (n1L + 1) + j - 1 ;
        elmntp->nps[2] = gp.N + (i-1) * (n1L + 1) + j ;
```

```
                index++ ;
                elmntp = alloc_elm(index,eltype) ;
                elmntp->nps[2] = gp.N + i * (nlL + 1) + j - 1 ;
                elmntp->nps[0] = gp.N + i * (nlL + 1) + j ;
                elmntp->nps[1] = gp.N + (i-1) * (nlL + 1) + j ;
            }
            if(index == N.elms-1)
                elmntp->nextelp = NULL ;
            break ;

    }
    return(0) ;
}

struct node *alloc_node(index,x,y)
int       index ;
double    x, y ;

{
    struct node   *nodep ;
    int  k ;

    nodep = gp.N + index ;
    nodep->n = index + 1  ;
    nodep->i = index  ;
    nodep->fxc = 0 ;
    gp.iptrs[index] = nodep ;

    nodep->x[0] = x ;
    nodep->x[1] = y ;
    for(k=0;k<N.params;k++)
        nodep->p[k] = defndp ;
    for(k=0;k<N.vars;k++)
        nodep->u[k] = 0.00 ;
    nodep->nextnp = nodep + 1 ;
    nodevalues(nodep) ;
    return(nodep) ;
}

struct element          *alloc_elm(index, eltype)
int       index, eltype ;

{
    int   k ;
    struct element  *elmntp ;

    elmntp = gp.El + index ;
    elmntp->n = index + 1 ;
    elmntp->vtype = eltype;
    elmntp->gtype = eltype;
    elmntp->nnds =  nsf(elmntp->vtype) ;
    if(eltype == 129) {
        elmntp->gtype = 111;
        elmntp->nnds = 4 ;
    }
    if(eltype == 139) {
        elmntp->gtype = 111;
        elmntp->nnds = 6 ;
    }
    for(k=0;k<N.params;k++)
        elmntp->p[k] = defelp ;
    elmntp->matrices = NULL ;
    elmntp->nextelp = elmntp + 1 ;
    elvalues(elmntp) ;

    return(elmntp) ;
}

struct belement         *alloc_belm(index, eltype)
int                     index, eltype ;

{
    int                 j, k ;
    struct belement     *belmntp ;

    belmntp = gp.B + index ;
    belmntp->n = index + 1 ;
    belmntp->vtype = eltype;
    belmntp->gtype = eltype;
    belmntp->nnds =  nsf(belmntp->vtype) ;
    for(k=0;k<N.bparams;k++)
```

```
        belmntp->p[k] = defelp ;
    for(j=0;j<N.vars;j++) {
        belmntp->bcs[j] = 0 ;
    }
    belmntp->matrices = NULL ;
    belmntp->nextbelp = belmntp + 1 ;

    return(belmntp) ;
}

int      map_mesh(theblock,maptype,rownum,colnum)
int      maptype, rownum, colnum ;
struct blockmap   *theblock ;

{
    struct node   *np ;
    struct gausspts    g ;
    struct shapefuncs  sf ;
    int       i, j ;

    np = gp.N ;
    g.z = 0.0 ;
    g.w = -1.0 ;
    for(i=0;i<N.nodes;i++) {
        g.x = np->x[0] - 2.0 * colnum ;
        g.y = np->x[1] - 2.0 * rownum ;
        if(get_shape(maptype,&g,&sf) == 0) {
            np->x[0] = 0.0 ;
            np->x[1] = 0.0 ;
            for(j=0;j<sf.dof;j++) {
                np->x[0] += sf.f[j] * theblock->xnodes[j] ;
                np->x[1] += sf.f[j] * theblock->ynodes[j] ;
            }
        }
        np = np->nextnp ;
    }
    return(0) ;
}
```

festuff.c

```
#include "fe.h"

extern struct control  M ;
extern double          UM ;

int  get_shape(el_type,x,psf)
int                     el_type ;
struct gausspts         *x ;
struct shapefuncs       *psf ;

{
     int  err ;
     switch (el_type) {


          case 111 :
               err = shap_ls1(x,psf) ;
               break ;

          case 119 :
               err = SUID_ls1(x,psf) ;
               break ;

          case 129 :
               err = shcu3_ls2(x,psf) ;
               break ;

          case 139 :
               err = shapcu_ls3(x,psf) ;
               break ;

     }
     return(err) ;
}


int  nsf(el_type)
int  el_type ;

{
     struct gausspts         x ;
     struct shapefuncs       psf ;
     int  err ;

     x.x = 100.0 ;
     x.y = 100.0 ;
     x.z = 100.0 ;
     err = get_shape(el_type,&x,&psf) ;
     return(err) ;
}


int  nbounds(el_type)
int  el_type ;

{
     int  err ;

     err = 0 ;
     switch (el_type) {


          case 111 :
               err = 2 ;
               break ;

          case 119 :
               err = 2 ;
               break ;

          case 129 :
               err = 2 ;
               break ;

          case 139 :
               err = 2 ;
```

```
                break ;
        }
    return(err) ;
}

int       get_nlcs(el_type,k,r,s)
int       k, el_type ;
double    *r, *s ;

{
    double   athird = 0.33333333333 ;

    switch (el_type) {

        case 111 :
            switch (k) {
                case 0 :
                    *r = 1.0 ;
                    *s = 1.0 ;
                    break ;
                case 1 :
                    *r = -1.0 ;
                    *s = 1.0 ;
                    break ;
            }
            return(2) ;

        }
        return(0) ;
}


int shap_1sl(x,p)
struct gausspts          *x ;
struct shapefuncs        *p ;

{
    double   r,mr,pr,t ;

    r = x->x ;
    if( (r<-1.0) || (r>1.0) ) return(2) ;

    t = 0.5 ;
    mr = 1.0 - r ;
    pr = 1.0 + r ;

    p->dof = 2 ;

    p->f[0] =  t * mr ;
    p->f[1] =  t * pr ;

    if(x->w < 0.0)
        return(0) ;

    p->dfdr[0] = -0.5 ;
    p->dfdr[1] =  0.5 ;

    return(0) ;
}

int  SU1D_1sl(x,p)
struct gausspts          *x ;
struct shapefuncs        *p ;

{
    double   r,mr,pr,t ;

    r = x->x ;
    if( (r<-1.0) || (r>1.0) ) return(2) ;

    t = 0.5 ;
    mr = 1.0 - r ;
    pr = 1.0 + r ;

    p->dof = 2 ;

    p->f[0] = -0.5 * UW ;
    p->f[1] =  0.5 * UW ;

    if(x->w < 0.0)
        return(0) ;
    p->dfdr[0] =  2 * UW * r ;
    p->dfdr[1] = -2 * UW * r;

    return(0) ;
}
int  shap_ls2(x,p)
struct gausspts          *x ;
struct shapefuncs        *p ;

{
    double   r,mr,pr,t ;

    r = x->x ;
    if( (r<-1.0) || (r>1.0) ) return(3) ;

    t = 0.5 ;
    mr = 1.0 - r ;
    pr = 1.0 + r ;

    p->dof = 3 ;

    p->f[0] = -t * r * mr ;
    p->f[1] = pr * mr ;
    p->f[2] = t * pr * r ;

    if(x->w < 0.0)
        return(0) ;

    p->dfdr[0] = -t * (1.0 - 2.0*r) ;
    p->dfdr[1] = -2.0 * r  ;
    p->dfdr[2] = t * (1.0 + 2.0*r) ;

    return(0) ;
}
int  shap_ls3(x,p)
struct gausspts          *x ;
struct shapefuncs        *p ;

{
    double   r,rm,rp,r3m,r3p,t ;

    r = x->x ;
    if( (r<-1.0) || (r>1.0) ) return(4) ;

    t = 9.0/144.0  ;
    rm = r - 1.0 ;
    rp = 1.0 + r ;
    r3m = 3.0 * r - 1.0 ;
    r3p = 3.0 * r + 1.0 ;

    p->dof = 4 ;

    p->f[0] = -t * rm * r3p * r3m ;
    p->f[1] = 9.0 * t * rp * rm * r3m ;
    p->f[2] = -9.0 * t * rp * rm * r3p ;
    p->f[3] = t * rp * r3p * r3m ;

    if(x->w < 0.0)
        return(0) ;

    p->dfdr[0] = -t * ( r3p * r3m + 3.0*rm*r3m + 3.0*rm*r3p) ;
    p->dfdr[1] = 9.0 * t * ( 3.0*rp * rm + rm*r3m + rm*r3p)  ;
    p->dfdr[2] = -9.0 * t * ( 3.0*rp * rm + rm*r3p + rp*r3p) ;
    p->dfdr[3] = t * ( r3p * r3m + 3.0*rm*r3m + 3.0*rm*r3p) ;

    return(0) ;
}
```

feice.c

```
#include "fe.h"

#define GRAV  9.81
extern struct control  N ;
extern struct pointers gp;


int       icejam(xtoe,xlen,kice)
double    xtoe,xlen,kice ;

{
     struct node          *np0, *npp, *npm ;
     double               eta, zeta, mu, slope, d, k, jelev ;
     int i ;

     jelev = -1 ;

     /* calcluations within the jam */

     npm = gp.N ;
     np0 = npm -> nextnp ;
     npp = np0 -> nextnp ;

     for (i=1;i<N.nodes-1;i++){
          if ((np0->x[0])<=xtoe && (np0->x[0])>=(xtoe-xlen)) {
               /* slope = Fy/Fx */
               slope = (npm->p[0] - npp->p[0])/(npp->x[0] - npm->x[0]) ;
               /* composite roughness */
               k = np0->p[1] ;
               k = pow(((pow(kice,4)+pow(k,4))/2),.25) ;
               /* d = depth to phreatic surface in equilibrium section of
                    fully-developed jam */
               zeta =   pow(( np0->u[1]/np0->p[2] * pow(k,(1./6.)) /sqrt(GRAV*slope)),.6) / slope/np0->p[2];
               mu =1.0 ;
               eta = 0.38 * zeta + 5.75/mu * (1 + sqrt(1+0.07*mu*zeta*pow(kice/k,.25))) ;
               d = eta * slope * np0->p[2] ;
               /* new area = d * width */
               np0->u[0] = d * np0->p[2] ;
               if(jelev<0) {
                    jelev= np0->p[0] + d ;
                    printf("\nk = %g\nzeta = %g\neta = %g\nd = %g\n",np0->p[1],zeta,eta,d);
               }
          }
          npm = np0 ;
          np0 = npp ;
          npp = npp -> nextnp ;
     }

     /* calcluations upstream of the jam */

     np0 = gp.N ;

     for (i=1;i<N.nodes;i++){
          if (np0->x[0]<=(xtoe-xlen)) {
               if(((np0->u[0]/np0->p[2]) + np0->p[0]) < jelev)
                    np0->u[0] = (jelev - np0->p[0]) * np0->p[2] ;
          }
          else break ;

          np0 = np0 -> nextnp ;
     }

     return(0) ;
}
```

fewindow.c

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include "fe.h"

Display *ws;
Window  og, rw;
GC  theGC;
XEvent event;
XSizeHints   hint;
XPoint   *Pfirst;
int  scr;
unsigned long     fg, bg;

char name[] = {"St. Venant"};

extern struct control N;
extern struct pointers gp;

struct element    *theElement;
struct node   *theNode;
struct belement   *theBoundEl;

double   winfactor, horigin, hscale, vorigin, vscale, vlorigin, vlscale;
double   ymin, ymax, xmin, xmax ;
int      wtp;

initXwindow(argc,argv)
int  argc;
char      **argv;
{
     int i, x, y;
     unsigned int d,w,h,bw;
     char text[10];
     int done;

     ws = XOpenDisplay("");
     scr = DefaultScreen(ws);
     bg = WhitePixel(ws, scr);
     fg = BlackPixel(ws,scr);

     hint.x = 50;
     hint.y = 50;
     hint.width = 900;
     hint.height = 650;
     hint.flags = PPosition | PSize;

     og = XCreateSimpleWindow(ws,DefaultRootWindow(ws),hint.x, hint.y, hint.width, hint.height,5, fg, bg );
     XSetStandardProperties(ws, og, name, name,None, argv, argc, &hint);

     theGC = XCreateGC(ws, og, 0, 0);
     XSetBackground(ws, theGC, bg);
     XSetForeground(ws, theGC, fg);
     XSelectInput(ws, og, ExposureMask);

     XMapRaised(ws, og);

     XNextEvent(ws,&event);

     Pfirst = NULL;
     winfactor = 0.65 ;
     wtp = 3;
     return(0);
}

killXwindow()

{
     XFreeGC(ws,theGC);
     XDestroyWindow(ws,og);
     XCloseDisplay(ws);
     if(Pfirst != NULL)
          free(Pfirst);

     return(0);
}
```

```
DrawDoubleNum(d,x,y)
double    d;
short     x, y;

{
     char      thenumstr[20];

     sprintf(thenumstr,"%9.2f",d);
     XDrawImageString(ws,og,theGC,x,y,thenumstr,strlen(thenumstr));

     return(0);
}

ScaleXwindow(scalecode)
int  scalecode;

{
     int  i, x, y, tm, bm, lm, rm, yw, ylw;
     unsigned int  d,w,h,bw;
     double   ylmin, ylmax, elev;
     struct node   *np;

     tm = 25;
     rm = 25;
     lm = 100;
     bm = 50;

     if(scalecode == 0) {
          np = gp.N;
          xmax = xmin = np->x[0];

          for(i=1;i<N.nodes;i++) {
               np = np->nextnp;
               if(np->x[0] > xmax) xmax = np->x[0];
               if(np->x[0] < xmin) xmin = np->x[0];
          }
     }

     if(scalecode == 1) {
          printf("Input xmin, xmax\n");
          scanf(" %lf %lf",&xmin,&xmax);
     }

     np = gp.N;
     ymin = ylmin = 9.999e20;
     ymax = ylmax = -9.999e20;

     for(i=0;i<N.nodes;i++) {
          if((np->x[0] > xmin) && (np->x[0] < xmax)) {
               if(np->x[wtp] > ylmax) ylmax = np->x[wtp];
               if(np->x[wtp] < ylmin) ylmin = np->x[wtp];
               elev = np->p[0]+(np->u[0]/np->p[2]);
               if(elev > ymax) ymax = elev;
               if(np->p[0] < ymin) ymin = np->p[0];
          }
          np = np->nextnp;
     }

     ylmax += (ylmax+ylmin);
     if(ylmax==0.)
          ylmax = 100. ;
     ylmin = 0.0 ;

     XGetGeometry(ws,og,&rw,&x,&y,&w,&h,&bw,&d);

     yw = (h-(tm+bm))*winfactor;
     ylw = h - (tm+bm+yw);

     hscale = (w-(lm+rm))/(xmax-xmin);
     horigin = (xmin+xmax)/2 - (w+lm-rm)/hscale/2;

     vscale = yw/(ymin-ymax);
     vorigin = (ymin+ymax)/2 - (h+tm+ylw-bm)/vscale/2;

     vlscale = ylw/(ylmin-ylmax);
     vlorigin = (ylmin+ylmax)/2 - (h+tm-yw-bm)/vlscale/2;
     if(Pfirst != NULL)
          free(Pfirst);

     Pfirst = (XPoint *) calloc(N.nodes,sizeof(XPoint));
     XClearWindow(ws,og);
```

```
    DrawBed();
    DrawElev();

    DrawDoubleNum(xmin,lm,h-bm+30);
    DrawDoubleNum(xmax,w-rm-80,h-bm+30);
    DrawDoubleNum(ymin,-10,h-bm);
    DrawDoubleNum(ymax,-10,h-bm-yw);
    DrawDoubleNum(y1min,-10,tm+y1w-30);
    DrawDoubleNum(y1max,-10,tm);
    XFlush(ws);

    return(0);
}

ScalePoint(pptr,x,y)
XPoint   *pptr;
double    x, y;

{
     pptr->x = (short) (((x-horigin)*hscale)+0.5) ;
     pptr->y = (short) (((y-vorigin)*vscale)+0.5) ;
     return(0) ;
}

Scale1Point(pptr,x,y)
XPoint   *pptr;
double    x, y;

{
     pptr->x = (short) (((x-horigin)*hscale)+0.5) ;
     pptr->y = (short) (((y-v1origin)*v1scale)+0.5) ;
     return(0) ;
}

DrawBed()

{
     int  i;
     XPoint   *theP ;
     struct node   *np;

     theP = Pfirst;
     np = gp.N;

     for(i=0;i<N.nodes;i++) {
          ScalePoint(theP++,np->x[0],np->p[0]);
          np = np->nextnp;
     }

     XDrawLines(ws,og,theGC,Pfirst,N.nodes,CoordModeOrigin);

     return(0);
}

DrawElev()

{
     int  i;
     XPoint   *theP ;
     struct node   *np;

     theP = Pfirst;
     np = gp.N;

     for(i=0;i<N.nodes;i++) {
          ScalePoint(theP++,np->x[0],np->p[0]+np->u[0]/np->p[2]);
          np = np->nextnp;
     }

     XDrawLines(ws,og,theGC,Pfirst,N.nodes,CoordModeOrigin);

     theP = Pfirst;
     np = gp.N;

     for(i=0;i<N.nodes;i++) {
          Scale1Point(theP++,np->x[0],np->x[wtp]);
          np = np->nextnp;
     }

     XDrawLines(ws,og,theGC,Pfirst,N.nodes,CoordModeOrigin);
```

```
        theP = Pfirst;
        np = gp.N;

        for(i=0;i<N.nodes;i++) {
             ScalePoint(theP++,np->x[0],ymax/* + (ymax -ymin)*0.05*/);
             np = np->nextnp;
        }

        XDrawPoints(ws,og,theGC,Pfirst,N.nodes,CoordModeOrigin);

        XFlush(ws);
        return(0);
    }

    SetAuxPlot(n)
    int      n;

    {
        wtp = n;
        return(0);
    }
```

APPENDIX D

Coefficients in the Linear Stability Analysis

D.1. Introduction

For the various finite element schemes, the equivalent finite difference temporal and spatial derivatives may be defined as:

$$\psi = \theta\left(\frac{\Psi_{j-1}^{n+1} + 4\Psi_{j}^{n+1} + \Psi_{j+1}^{n+1}}{6}\right) + \left(1-\theta\right)\left(\frac{\Psi_{j-1}^{n} + 4\Psi_{j}^{n} + \Psi_{j+1}^{n}}{6\Delta t}\right) \qquad [D.1]$$

$$\frac{\partial\psi}{\partial t} = \frac{\Psi_{j-1}^{n+1} + 4\Psi_{j}^{n+1} + \Psi_{j+1}^{n+1}}{6\Delta t} - \frac{\Psi_{j-1}^{n} + 4\Psi_{j}^{n} + \Psi_{j+1}^{n}}{6\Delta t} \qquad [D.2]$$

$$\frac{\partial\psi}{\partial x} = \theta\left(\frac{\Psi_{j+1}^{n+1} - \Psi_{j-1}^{n+1}}{2\Delta x}\right) + \left(1-\theta\right)\left(\frac{\Psi_{j+1}^{n} - \Psi_{j-1}^{n}}{2\Delta x}\right) \qquad [D.3]$$

$$\frac{\partial^2\psi}{\partial x^2} = \theta\left(\frac{\Psi_{j-1}^{n+1} - 2\Psi_{j}^{n+1} + \Psi_{j+1}^{n+1}}{\Delta x^2}\right) + \left(1-\theta\right)\left(\frac{\Psi_{j-1}^{n} - 2\Psi_{j}^{n} + \Psi_{j+1}^{n}}{\Delta x^2}\right) \qquad [D.4]$$

$$\frac{\partial^2\psi}{\partial x\partial t} = \left(\frac{\Psi_{j+1}^{n+1} - \Psi_{j-1}^{n+1}}{2\Delta x\Delta t}\right) - \left(\frac{\Psi_{j+1}^{n} - \Psi_{j-1}^{n}}{2\Delta x\Delta t}\right) \qquad [D.5]$$

D.2 The Bubnov-Galerkin Method

For the Bubnov-Galerkin scheme substitution gives the following values for the coefficients in equation [4.10]:

$$a_{1,1} = \frac{1}{6}; \qquad a_{2,1} = -\,a_{qa}\,\theta\,\frac{\Delta t}{2\Delta x} \qquad [D.6]$$

$$a_{1,2} = \frac{2}{3}; \qquad a_{2,2} = 0 \qquad [D.7]$$

$$a_{1,3} = \frac{1}{6}; \qquad a_{2,3} = a_{qa}\,\theta\,\frac{\Delta t}{2\Delta x} \qquad [D.8]$$

$$a_{1,4} = -\theta\frac{\Delta t}{2\Delta x}; \qquad a_{2,4} = \frac{1}{6} - a_{qq}\theta\frac{\Delta t}{2\Delta x} \tag*{[D.9]}$$

$$a_{1,5} = 0; \qquad a_{2,5} = \frac{2}{3} \tag*{[D.10]}$$

$$a_{1,6} = \theta\frac{\Delta t}{2\Delta x} \qquad a_{2,6} = \frac{1}{6} + a_{qq}\theta\frac{\Delta t}{2\Delta x} \tag*{[D.11]}$$

$$a_{1,7} = -\frac{1}{6}; \qquad a_{2,7} = -a_{qa}\left(1-\theta\right)\frac{\Delta t}{2\Delta x} \tag*{[D.12]}$$

$$a_{1,8} = -\frac{2}{3}; \qquad a_{2,8} = 0 \tag*{[D.13]}$$

$$a_{1,9} = -\frac{1}{6}; \qquad a_{2,9} = a_{qa}\left(1-\theta\right)\frac{\Delta t}{2\Delta x} \tag*{[D.14]}$$

$$a_{1,10} = -a_{qa}\left(1-\theta\right)\frac{\Delta t}{2\Delta x}; \qquad a_{2,10} = -\frac{1}{6} - a_{qq}\left(1-\theta\right)\frac{\Delta t}{2\Delta x}; \tag*{[D.15]}$$

$$a_{1,11} = 0; \qquad a_{2,11} = \frac{2}{3} \tag*{[D.16]}$$

$$a_{1,12} = \left(1-\theta\right)\frac{\Delta t}{2\Delta x}; \qquad a_{2,12} = -\frac{1}{6} + a_{qq}\left(1-\theta\right)\frac{\Delta t}{2\Delta x} \tag*{[D.17]}$$

where:

$$a_{qa} \equiv \left(1 - \mathbf{F}^2\right) \tag*{[D.18]}$$

and,

$$a_{qq} \equiv 2\mathbf{F} \tag*{[D.19]}$$

D.3 The Taylor-Galerkin Method

For the Taylor-Galerkin scheme substitution gives the following values for the coefficients in equation [4.10]:

$$a_{1,1} = \frac{1}{6}; \qquad a_{2,1} = 0 \tag*{[D.20]}$$

$$a_{1,2} = \frac{2}{3}; \qquad a_{2,2} = 0 \qquad [D.21]$$

$$a_{1,3} = \frac{1}{6}; \qquad a_{2,3} = 0 \qquad [D.22]$$

$$a_{1,4} = 0; \qquad a_{2,4} = \frac{1}{6} \qquad [D.23]$$

$$a_{1,5} = 0; \qquad a_{2,5} = \frac{2}{3} \qquad [D.24]$$

$$a_{1,6} = 0 \qquad a_{2,6} = \frac{1}{6} \qquad [D.25]$$

$$a_{1,7} = -\frac{1}{6} - \frac{\Delta t}{2\Delta x} - a_{qa}\frac{\Delta t^2}{2\Delta x^2}; \qquad a_{2,7} = -a_{qa}\frac{\Delta t}{2\Delta x} - a_{qa}\,a_{qq}\frac{\Delta t^2}{2\Delta x^2} \qquad [D.26]$$

$$a_{1,8} = -\frac{2}{3} + a_{qa}\frac{\Delta t^2}{\Delta x^2}; \qquad a_{2,8} = a_{qa}\,a_{qq}\frac{\Delta t^2}{\Delta x^2} \qquad [D.27]$$

$$a_{1,9} = -\frac{1}{6} + \frac{\Delta t}{2\Delta x} - a_{qa}\frac{\Delta t^2}{2\Delta x^2}; \qquad a_{2,9} = a_{qa}\frac{\Delta t}{2\Delta x} - a_{qa}\,a_{qq}\frac{\Delta t^2}{2\Delta x^2} \qquad [D.28]$$

$$a_{1,10} = -\frac{\Delta t}{2\Delta x} - a_{qq}\frac{\Delta t^2}{2\Delta x^2}; \qquad a_{2,10} = -\frac{1}{6} - a_{qq}\frac{\Delta t}{2\Delta x} - \left(a_{qa} + a_{qq}^2\right)\frac{\Delta t^2}{2\Delta x^2} \qquad [D.29]$$

$$a_{1,11} = a_{qq}\frac{\Delta t^2}{\Delta x^2}; \qquad a_{2,11} = -\frac{2}{3} + \left(a_{qa} + a_{qq}^2\right)\frac{\Delta t^2}{\Delta x^2} \qquad [D.30]$$

$$a_{1,12} = \frac{\Delta t}{2\Delta x} - a_{qq}\frac{\Delta t^2}{2\Delta x^2}; \qquad a_{2,12} = -\frac{1}{6} + a_{qq}\frac{\Delta t}{2\Delta x} - \left(a_{qa} + a_{qq}^2\right)\frac{\Delta t^2}{2\Delta x^2} \qquad [D.31]$$

D.4 The Petrov-Galerkin Method

For the Petrov-Galerkin scheme, the coefficients in equation [4.10] may be generalized to:

$$a_{1,1} = \frac{1}{6} + \frac{\omega\, w_{aa}}{4} - \frac{\omega\, w_{aq}\, a_{qa}}{2}\frac{\theta\,\Delta t}{\Delta x};$$

$$a_{2,1} = -\frac{\theta\,\Delta t}{2\Delta x}\left(a_{qa} + \omega\, w_{qq}\, a_{qa}\right) + \frac{\omega\, w_{qa}}{4} \qquad [D.32]$$

$$a_{1,2} = \frac{2}{3} + \omega\, w_{aq}\, a_{qa} \frac{\theta\, \Delta t}{\Delta x};$$

$$a_{2,2} = \left(w_{qq}\, a_{qa}\right) \frac{\omega\, \theta\, \Delta t}{\Delta x} \qquad [D.33]$$

$$a_{1,3} = \frac{1}{6} - \frac{\omega\, w_{aa}}{4} - \frac{\omega\, w_{aq}\, a_{qa}}{2} \frac{\theta\, \Delta t}{\Delta x};$$

$$a_{2,3} = \frac{\theta\, \Delta t}{2\Delta x} \left(a_{qa} - \omega\, w_{qq}\, a_{qa}\right) - \frac{\omega\, w_{qa}}{4} \qquad [D.34]$$

$$a_{1,4} = -\frac{\theta\, \Delta t}{2\Delta x} \left(1 + \omega\, w_{aa} + \omega\, w_{aq}\, a_{qq}\right) - \frac{\omega\, w_{aq}}{4};$$

$$a_{2,4} = \frac{1}{6} + \frac{\omega\, w_{qq}}{4} - \frac{\theta\, \Delta t}{2\Delta x} \left(a_{qq} + \omega\, w_{qa} + \omega\, w_{qq}\, a_{qq}\right) \qquad [D.35]$$

$$a_{1,5} = \frac{\omega\, \theta\, \Delta t}{\Delta x} \left(w_{aa} + w_{aq}\, a_{qq}\right);$$

$$a_{2,5} = \frac{2}{3} + \frac{\omega\, \theta\, \Delta t}{2\Delta x} \left(w_{qa} + w_{qq}\, a_{qq}\right) \qquad [D.36]$$

$$a_{1,6} = \frac{\theta\, \Delta t}{2\Delta x} \left(1 - \omega\, w_{aa} - \omega\, w_{aq}\, a_{qq}\right) + \frac{\omega\, w_{aq}}{4};$$

$$a_{2,6} = \frac{1}{6} - \frac{\omega\, w_{qq}}{4} + \frac{\theta\, \Delta t}{2\Delta x} \left(a_{qq} - \omega\, w_{qa} - \omega\, w_{qq}\, a_{qq}\right) \qquad [D.37]$$

$$a_{1,7} = -\frac{1}{6} - \frac{\omega\, w_{aa}}{4} - \frac{\omega\, w_{aq}\, a_{qa}}{2} \frac{\left(1-\theta\right) \Delta t}{\Delta x};$$

$$a_{2,7} = -\frac{\left(1-\theta\right) \Delta t}{2\Delta x} \left(a_{qa} + \omega\, w_{qq}\, a_{qa}\right) - \frac{\omega\, w_{qa}}{4} \qquad [D.38]$$

$$a_{1,8} = -\frac{2}{3} + \omega\, w_{aq}\, a_{qa} \frac{\left(1-\theta\right) \Delta t}{\Delta x};$$

$$a_{2,8} = \left(w_{qq}\, a_{qa}\right) \frac{\omega \left(1-\theta\right) \Delta t}{\Delta x} \qquad [D.39]$$

$$a_{1,9} = -\frac{1}{6} + \frac{\omega\, w_{aa}}{4} - \frac{\omega\, w_{aq}\, a_{qa}}{2} \frac{\left(1-\theta\right) \Delta t}{\Delta x};$$

$$a_{2,9} = \frac{(1-\theta)\Delta t}{2\Delta x}\left(a_{qa} - \omega w_{qq} a_{qa}\right) + \frac{\omega w_{qa}}{4} \qquad [D.40]$$

$$a_{1,10} = -\frac{(1-\theta)\Delta t}{2\Delta x}\left(1 + \omega w_{aa} + \omega w_{aq} a_{qq}\right) - \frac{\omega w_{aq}}{4} \; ;$$

$$a_{2,10} = -\frac{1}{6} - \frac{\omega w_{qq}}{4} - \frac{(1-\theta)\Delta t}{2\Delta x}\left(a_{qq} + \omega w_{qa} + \omega w_{qq} a_{qq}\right) \qquad [D.41]$$

$$a_{1,11} = \frac{\omega(1-\theta)\Delta t}{\Delta x}\left(w_{aa} + w_{aq} a_{qq}\right);$$

$$a_{2,11} = -\frac{2}{3} + \frac{\omega(1-\theta)\Delta t}{2\Delta x}\left(w_{qa} + w_{qq} a_{qq}\right) \qquad [D.42]$$

$$a_{1,12} = \frac{(1-\theta)\Delta t}{2\Delta x}\left(1 - \omega w_{aa} - \omega w_{aq} a_{qq}\right) + \frac{\omega w_{aq}}{4} \; ;$$

$$a_{2,12} = -\frac{1}{6} + \frac{\omega w_{qq}}{4} + \frac{(1-\theta)\Delta t}{2\Delta x}\left(a_{qq} - \omega w_{qa} - \omega w_{qq} a_{qq}\right) \qquad [D.43]$$

D.5 The Box Finite Difference Scheme

In the Box Finite Difference scheme, the temporal and spatial derivatives may be defined as shown in Chapter 3:

$$\frac{\partial \psi}{\partial t} = \frac{\Psi_j^{n+1} - \Psi_j^n}{2\Delta t} + \frac{\Psi_{j+1}^{n+1} - \Psi_{j+1}^n}{2\Delta t} \qquad [3.18]$$

$$\frac{\partial \psi}{\partial x} = \theta\left(\frac{\Psi_{j+1}^{n+1} - \Psi_j^{n+1}}{\Delta x}\right) + (1-\theta)\left(\frac{\Psi_{j+1}^n - \Psi_j^n}{\Delta x}\right) \qquad [3.19]$$

Substitution gives the following values for the coefficients in equation [4.10]:

$$a_{1,1} = 0\,; \qquad a_{2,1} = 0 \qquad [D.44]$$

$$a_{1,2} = \frac{1}{2}\,; \qquad a_{2,2} = -\,a_{qa}\,\theta\,\frac{\Delta t}{\Delta x} \qquad [D.45]$$

$$a_{1,3} = \frac{1}{2}; \qquad a_{2,3} = a_{qa}\,\theta\,\frac{\Delta t}{\Delta x} \qquad [D.46]$$

$$a_{1,4} = 0; \qquad a_{2,4} = 0 \qquad [D.47]$$

$$a_{1,5} = -\theta\,\frac{\Delta t}{\Delta x}; \qquad a_{2,5} = \frac{1}{2} - a_{qq}\,\theta\,\frac{\Delta t}{\Delta x} \qquad [D.48]$$

$$a_{1,6} = \theta\,\frac{\Delta t}{\Delta x}; \qquad a_{2,6} = \frac{1}{2} + a_{qq}\,\theta\,\frac{\Delta t}{\Delta x} \qquad [D.49]$$

$$a_{1,7} = 0; \qquad a_{2,7} = 0 \qquad [D.50]$$

$$a_{1,8} = -\frac{1}{2}; \qquad a_{2,8} = -a_{qa}\left(1-\theta\right)\frac{\Delta t}{\Delta x} \qquad [D.51]$$

$$a_{1,9} = -\frac{1}{2}; \qquad a_{2,9} = a_{qa}\left(1-\theta\right)\frac{\Delta t}{\Delta x} \qquad [D.52]$$

$$a_{1,10} = 0; \qquad a_{2,10} = 0 \qquad [D.53]$$

$$a_{1,11} = -\left(1-\theta\right)\frac{\Delta t}{\Delta x}; \qquad a_{2,11} = -\frac{1}{2} - a_{qq}\left(1-\theta\right)\frac{\Delta t}{\Delta x} \qquad [D.54]$$

$$a_{1,12} = \left(1-\theta\right)\frac{\Delta t}{\Delta x}; \qquad a_{2,12} = -\frac{1}{2} + a_{qq}\left(1-\theta\right)\frac{\Delta t}{\Delta x} \qquad [D.55]$$